国家精品课程教材

高等学校规划教材

大学计算机基础

（第2版）

◎ 吴　宁　主编　◎ 崔舒宁　陈文革　李威威　编著
◎ 冯博琴　主审

電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn

国家精品课程教材
高等学校规划教材

大学计算机基础

（第2版）

主编　吴　宁

编著　崔舒宁　陈文革　李威威

主审　冯博琴

電子工業出版社

Publishing House of Electronics Industry

北京・BEIJING

内 容 简 介

本书是国家精品课程“大学计算机基础”的主教材，是在《大学计算机基础》的基础上修订而成的，此次再版，融入了两年教学实践的新体会，同时，对部分内容进行了调整和修改。

本书以“计算思维能力”培养为主线，强调“计算机基本工作原理”的理解和“问题求解思路”的建立。在组织架构上主要分为四个部分，共 8 章，主要内容包括计算与可计算性理论简述（引言部分）、计算机基础知识、微型计算机系统、计算机网络及应用、Visual Basic 程序设计、数据结构、算法分析与设计和综合案例。

本书各章均在起始处给出了该章的引言及教学目的，以供读者学习时参考。同时，还配有大量图示和例题，以便于对内容的理解。为方便教学，本书还免费提供电子课件，可以登录华信教育资源网（www.hxedu.com.cn）免费注册下载。

本书可作为普通高等学校非计算机专业“大学计算机基础”课程的教材，适用学时为 48～64 学时。目录中带有“*”的章节为可选内容，可根据具体情况选择讲授或作为课外开放性学习使用。

图书在版编目（CIP）数据

大学计算机基础 / 吴宁主编；崔舒宁，陈文革，李威威编著. —2 版. —北京：电子工业出版社，2013.8
高等学校规划教材

ISBN 978-7-121-15529-1

Ⅰ. ①大… Ⅱ. ①吴… ②崔… ③陈… ④李… Ⅲ.①电子计算机—高等学校—教材 Ⅳ. ①TP3

中国版本图书馆 CIP 数据核字（2013）第 169687 号

策划编辑：索蓉霞
责任编辑：郝黎明
印　　刷：
装　　订：
出版发行：电子工业出版社
　　北京市海淀区万寿路 173 信箱　　邮编：100036
开　　本：787×1 092　1/16　印张：17.75　字数：454.4 千字
印　　次：2013 年 8 月第 1 次印刷
定　　价：35.00 元

凡所购买电子工业出版社图书有缺损问题，请向购买书店调换。若书店售缺，请与本社发行部联系，联系及邮购电话：（010）88254888。

质量投诉请发邮件至 zlts@phei.com.cn，盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线：（010）88258888。

前　言

《大学计算机基础》第 1 版自 2011 年出版以来，已在我校使用两个学年。本书是在 2011 年基础上修订而成的。此次再版，仍保持了第 1 版教材的“以计算思维能力培养为主线，强调计算机基本工作原理理解和问题求解思路建立”这一宗旨，但融入了近两年来的教学体会及新的教学研究成果，对第 1 版中的部分内容进行了精简，适当加强了可计算性理论及操作系统的介绍。主要修订内容如下：

（1）按照以“计算思维”为切入点的培养需求，加入了计算与可计算性理论的描述，以帮助读者进一步了解哪类问题是计算机可计算的，哪些是不可计算的。

（2）加强了操作系统原理描述，特别是进程调度及存储器管理等内容。目的是使读者能更深入地理解多任务并发执行的原理，以及计算机中存储器系统的构成和管理模式。

（3）对原计算机网络部分的内容进行了修订。更新为：以互联网为主线介绍计算机网络的基本概念、工作原理和应用。同时，在有关网络安全技术的介绍上，更加具有条理性和逻辑性，不仅包括了网络安全技术原理，也包括了网络安全技术的实际应用。

（4）为了进一步提升学习者“利用计算机求解问题的能力”，将数据结构与算法设计分为两章，对内容进行了充实，新增了一些设计示例，以帮助读者理解相关内容。

本次再版，既考虑了目前全国高等学校入校新生在计算机基础知识方面的一般现状，更考虑了创新型人才必须具备利用计算机求解问题的能力这一需求。力求使读者通过本书的学习，能够在了解计算机基础知识的基础上，较为深入地理解微型计算机的基本工作原理，能够初步建立起利用计算机解决问题的思路、掌握求解问题的一般方法，并了解利用计算机解决问题的一般过程，以及了解哪类问题是计算机不可解决的。

全书在组织架构上主要分为四个部分，共 8 章。

四部分内容：一是计算与可计算性理论；二是计算机中的信息表示；三是微型计算机系统组成和基本工作原理；四是算法和数据结构设计和实现。

第 1 章为引论，主要介绍了计算与可计算性理论、计算工具的发展、计算机问题求解过程、当前计算机科学研究的前沿技术等。第 2 章～第 4 章为计算机基础知识、系统组成、操作系统和网络技术等系统平台知识。第 5 章～第 7 章保持了第 1 版的主要内容，以“建立利

用计算机解决问题的思路和方法”为宗旨，主要介绍利用 VB 程序语言进行数据结构和简单算法的设计及算法描述方法等。第 8 章仍然是一些综合案例设计。

本书配备同步修订的实验指导书大学计算机基础实验教材(Windows7+Office 2010)（第 2 版），新的实验指导教程也随着技术的发展及教学研究的成果做了一定的修订。为方便教学，本书还免费提供电子课件，可以登录华信教育资源网（www.hxedu.com.cn）注册下载。

本书由吴宁主编并统稿，参与编写的有吴宁（第 1～3 章）、陈文革（第 4 章）、崔舒宁（第 5～8 章）、李威威整理了附录并负责全书的校对；程向前和贾应智二位老师提供了部分案例。本书由首届国家级教学名师冯博琴教授主审，他为本书提出了许多宝贵的意见和建议，藉此表示衷心的感谢。

虽然经两年多的教学体会，使此次修订较第 1 版在内容安排、编写风格等方面都有一些改进，但作为一门新兴的课程，“大学计算机基础”的教学内容及相应教材的编写依然是难点，加之作者水平所限，书中错误和不妥之处在所难免，恳望师生不吝指正，十分感谢。作者 E-mail：wun@mail.xjtu.edu.cn。

编　者

于西安交通大学

2013 年 8 月

目　录

第1章 引 论

引言

1991年，美国施乐公司PARC研究中心的Mark Weiser在Scientific American上发表的题为“Computer for the 21th Century”的文章中，提出了“无处不在的计算（Ubiquitous Computing）”（又译为“普适计算”）的理念，开创了计算领域的第三次浪潮[1]。无处不在的计算设备，无处不在的网络和通信，彻底改变了人类数千年的生活习惯。人们希望通过无处不在的计算，能随时随地获得自己希望的服务，且不用关心这些服务是怎样得到的。由于提供这些服务或计算的重要载体是计算机，因此，现代信息社会的每个人，都需要了解计算机，学习计算机科学。而作为未来的工程计算机人员和科学家，更需要具备利用计算机解决相关问题的能力，以及判断什么样的问题可以由计算机解决的能力。作为全书的“引论”，本章将从什么是计算开始，简要介绍可计算性理论、计算机中的信息表示、基于计算机的问题求解过程，以及当前计算机科学研究的一些前沿技术。

教学目的

- 了解计算与可计算性。
- 理解图灵机模型及其工作过程。
- 了解计算机的发展。
- 了解计算机中的信息表示及信息的处理过程。
- 了解基于计算机的问题求解过程。
- 了解当前计算机科学研究的前沿技术。

1.1 计算与可计算性

今天，计算机已深入到生活的各个角落，几乎每个人都知道计算机能做很多事，并且似乎是什么事都能做。在各类关于机器人的影视作品中，人“机”大战也成为眩目的看点。计算机真的什么都能做吗？是否真的会出现地球被机器人统治的那一天呢？

答案是否定的。20世纪30年代，图灵（Turing）就已经证明了计算机能力的极限性，即存在计算机不能解决的问题。这类问题又称为不可计算问题。

作为学习计算机科学的入门教材，首先使读者了解“不是任何问题都是计算机可解的”这一概念是非常必要的。虽然在现代科学研究中，没有计算机是万万不能的，但计算机确实不是万能的。

[1] 计算的第一次浪潮是主机计算，人们通过字符终端共享主机计算；第二次浪潮是桌面计算时代，即个人计算机网络通信时代。

1.1.1 计算与计算科学

1．计算

计算的行为由来已久，考古研究说明，在远古时代，古人类就有了计算问题的需要和能力。人类最初的计算工具就是人类的双手，掰着指头数数就是最早的计算方法。一个人天生有十个指头，因此，十进制就成为人们最熟悉的进制计数法。

由于双手的局限性，人类开始学习用小木棍、石子、“结绳”等方法进行计算。英文中的计算一词“Calculation”来自拉丁文中的“Calculus”，其本意就是用于计算的小石子。在2000多年前古代中国人发明的算筹是世界上最早的计算工具，而具有十进制计数法和一整套计算口诀的算盘，则可以认为是最早的“数字计算机”，珠算口诀就是最早的体系化的算法。

但到底什么是“计算”？这个问题直到20世纪30年代人们才从哥德尔（K.Godel）、丘奇（A.Church）、艾伦·麦席森·图灵（A.M.Turing）等科学家的研究中弄清楚是计算的本质，更重要的是弄清楚了什么问题是可计算的，什么问题是不可计算的。

计算，抽象地讲，就是从一个符号串X变成另一个符号串Y的过程。例如：

（1）从符号串5+8−4变换为9，是进行了加减计算。

（2）将$X=a^2$变换为Y=2a，进行了微分计算。

（3）将一段英文（符号串X）翻译为中文（符号串Y），也是进行了计算。

从以上这几个简单示例可以看出，计算是按照一定的、有限的规则和步骤（算法），将输入转换为输出的过程。

从数学的角度，计算主要可以分为数值计算和符号推导两大类。数值计算包括各种算术运算、方程求解等，如上例的（1）和（2）；符号推导包括各种函数的等式或不等式的证明、几何命题证明等。上例中的（3）可以归为符号推导。

2．计算科学

随着计算机技术的发展，计算这个原本专门的数学概念已经泛化到了人类的整个知识领域，并成为一种普适的科学概念。计算不再仅是数学的基础技能，而且是整个自然科学的工具。今天，人类的研究领域越来越广，每个领域的研究都会涉及大量的计算。作为各门学科研究的基础，在现代计算工具的支撑下，计算也逐渐形成体系，成为一门独立的学科，即计算科学，它是涉及数学模型构建、数值分析方法及计算机实现方法的研究领域。

数学模型（Mathematical Model）是用数学符号、数学公式、程序、图形等对客观问题本质属性的抽象而又简洁的刻划，它或能解释某些客观现象，或能预测未来的发展规律，或能为控制某一现象的发展提供某种意义下的最优策略或较好策略。数学模型的建立首先需要对现实问题进行深入细致的观察和分析，抽象出各种关键因素并确立它们之间的关系，再灵活巧妙地利用各种数学知识，表述这些关系。例如，学生综合素质测评、股市变化分析和气象综合预测等问题，都可以建立成一个数学模型。通过研究对象特征，分析对象的内在规律，确定影响变化规律的因素，利用适当的数学工具，构造出各因素间的关系。

这种应用相关领域知识从客观问题中抽象、提炼出数学模型的过程就称为数学建模（Mathematical Modeling）。数学建模的核心是分析和抽象，即要能从复杂的客观现实中抽取出反映其变化规律的各种因素，并建立它们之间的关系。因此，建立数学模型的目的，就是为了便于分析和研究各种客观现象的运动规律，从而确立最佳的问题解决方案。

数值分析（Numerical Analysis）是关于数值近似算法的研究。古巴比伦人曾利用巴比伦泥板（数值分析的最早作品之一）来计算近似值，而不是精确值。引入数值分析的原因是因为在许多实际问题中，常常无法求得精确值，或是无法用有理数表示结果。数值分析的目的就是在合理误差范围内，求得满足精度要求的近似结果。数值分析方法在所有工程和科学领域中得到应用。例如，数值天气预报、太空船的轨迹计算、股票市值及其变异程度分析、保险公司的精算分析等。

计算机实现方法可以描述为利用计算机求解问题的方法。在模型建立的前提下，“计算机实现”的核心就是算法设计和程序设计。

虽然计算科学不完全等同于计算机科学，但计算科学，以及其他各学科的研究都离不开计算机。因此，“利用计算机分析和解决相关问题”的能力成为每一位科学工作者应具备的基本素质。这种能力又称为计算思维（Computational Thinking）[1] 能力。

计算机科学（Computer Science）是主要研究计算理论、计算机及信息处理的学科。半个多世纪来，计算机科学得到飞速发展和普及，作为现代科学体系的主要基石之一，它已逐渐超出一门单独学科的范围，演变为一种与社会、经济、能源、材料和健康等多个领域相结合的横向型科学技术。

有关计算机科学的详细定义有多种，但不论哪种定义，都强调了算法的研究。算法描述了解决某个特定问题的确切、无歧义、有限的动作序列。例如，按照某些准则获取一个年级中综合成绩最好的学生的过程，就是一种算法。遵照菜谱一步步做出一道好菜的过程，也是一种算法。计算机科学既要研究算法分析与设计理论，也同时要考虑如何在计算机上实现算法并解决实际问题。

1.1.2 可计算性理论

要利用计算机解决问题，或者说完成某项任务，首先需要使计算机明确按照什么样的方法和步骤工作，即确定算法。计算机能不能完成一项任务，取决于能不能设计出完成这项任务的算法。人类很早就已开始研究算法，我国的珠算口诀、求两个正整数的最大公约数所用的辗转相除法等，都可以称为算法。

可计算性理论（Computability Theory）是研究计算的一般性质的数学理论。由于计算的过程就是执行算法的过程，因此，可计算理论的中心课题就是将算法这一直观概念精确化，建立计算的数学模型，研究哪些是可计算的，哪些是不可计算的，以此揭示计算的实质。由于计算与算法联系在一起，因此，可计算性理论又称为算法理论。

直观上说，求解一类问题的算法是一组规则，这组规则条数有限，每一条都是可执行的（可操作的），并且这种操作性是绝对机械的，即不论何人何时对之进行操作，只要输入数据相同，其结果都是一样的。作为算法的一组规则，至少还应包含一条有关终止计算的

[1]“计算思维”是运用计算机科学的基础概念进行问题求解、系统设计，以及人类行为理解等涵盖计算机科学之广度的一系列思维活动。

条目。因此，从直观上看，算法具备的特征是有限性、可执行性、机械性、确定性和终止性。

在 20 世纪以前，对“算法”、“计算”这些概念似乎并不存在什么问题，人们普遍认为所有的问题都是有算法的，至少是一切数学命题都存在算法。但是 20 世纪初，人们发现有许多问题虽已经过长期研究，但仍然找不到算法，如希尔伯特第 10 问题及半群的字的问题等。于是人们开始怀疑，是否对某些问题根本就不存在算法？即存在不可计算的问题？

于是，数学家们开始了对算法概念及可计算性的精确化研究。1934 年，哥德尔（Godel）提出了一般递归函数的概念，并指出：凡算法可计算函数[1]都是一般递归函数，反之亦然。这一定义后来被称为埃尔布朗 • 哥德尔 • 克林定义。同年，丘奇证明了他提出的 λ 可定义函数与一般递归函数是等价的，并提出算法可计算函数等同于一般递归函数或 λ 可定义函数，即著名的“丘奇论点”。用一般递归函数虽给出了可计算函数的严格数学定义，但在具体的计算过程中，就某一步运算而言，选用什么初始函数和基本运算仍有不确定性。为消除所有的不确定性，图灵和 E • 波斯特各自独立地提出了抽象计算机的概念。图灵在他的“论可计算数及其在判定问题中的应用”（on Computable Numbers，with an Application to the Entscheidungsproblem）一文中，从一个全新的角度定义了可计算函数，他全面分析了人的计算过程，把计算归结为最简单、最基本、最确定的操作动作，从而用一种简单的方法来描述那种直观上具有机械性的基本计算程序，使任何机械（能行）的程序都可以归结为这些动作。这种简单的方法以一个抽象自动机概念为基础，并已证明：这种自动机能计算的函数是可计算函数，而这种自动机不能计算的函数则是不可计算的函数。这种抽象自动机被后人称为图灵机。

1937 年，图灵在他的“可计算性与 λ 可定义性”一文中证明了图灵机可计算函数与 λ 可定义函数是等价的，从而拓广了丘奇论点，得出：算法（能行）可计算函数等同于一般递归函数或 λ 可定义函数或图灵机可计算函数，即“丘奇—图灵论点”，相当完善地解决了可计算函数的精确定义问题，对数理逻辑的发展起了巨大的推动作用。

虽然已经证明可计算函数就是图灵机可计算函数，而图灵机可计算函数是递归可枚举函数。但这样的定义或描述对初学者来讲似乎显得有点过于高深。事实上，由于图灵机与计算机可以相互模拟（见 3.3.3 节），所以，对“可计算性”问题我们也可以通俗地描述为就是可以用计算机来解决的问题。从广义上讲，如“请为我做一个汉堡”这样的问题是无法用计算机来解决的（至少目前还不行）。计算机本身的优势在于数值计算（很多如文字识别，图像处理等非数值问题都可以通过转化为数值问题），能够用计算机解决的问题一定是“可以在确定、有限步骤内被解决的问题”，即有确定算法。像哥德巴赫猜想这样的问题就不属于“可计算问题”之列，因为计算机没有办法给出数学意义上的证明。

有关可计算性理论的深入描述因篇幅及课程性质所限，不再在此做进一步的讨论。本节讨论可计算性问题的目的，是希望读者能初步了解：[计算机不可能解决世界上所有的问题。其“不可解决性”反映在两个方面：一是不能在有限步骤内被解决；二是虽然有可能解决，但因过于复杂而不能在可接受的时间内解决]（所有机械装置都存在复杂性的临界点）。关于后者见 7.3 节算法的复杂性评价。

分析某个问题的可计算性意义重大，它可以使人们不必把时间浪费在不可能解决的问

[1] 可计算函数定义：能够在抽象计算机上编出程序计算其值的函数。

题上，而将精力用于可以解决的问题中。

1.1.3 图灵机模型

1．图灵机模型

图灵机（Turing Machine）是图灵 1936 年提出的一种抽象计算模型，其基本思想：用机器来模拟人用纸和笔进行数学运算的过程。

图灵将人的计算过程看作以下两个简单的动作：

（1）在纸上写上或擦除某个符号；

（2）将注意力从纸上的一个位置移动到另一个位置，而人每一次的下一步动作走向依赖于人当前所关注的纸上某个位置的符号及人当前的思维状态。

为了模拟人的这种运算过程，图灵构造出一台假想的（抽象的）机器（图 1-1），该机器由以下几个部分组成：

- 一条无限长的纸带 Type。纸带被划分成一个个连续的方格。每个格子上可包含一个来自有限字母表的符号，字母表中有一个特殊的符号表示空白。纸带上的格子从左到右依此被编号为 0，1，2，…，纸带的右端可以无限延长。
- 一个读/写头 Head（图 1-1 中间的大盒子）。读/写头内部包含了一组固定的状态（盒子上的方块）和程序。该读/写头可以在纸带上左右移动，它能读出当前所指的格子上的符号，并能改变当前格子上的符号。
- 一套控制规则 Table（即程序）。Table 包括当前读/写头的内部状态，输入数值，输出数值，下一时刻的内部状态。在每个时刻，读/写头都从当前纸带上读入一个方格信息。根据当前机器所处的状态及读/写头所读入的格子上的符号来确定读/写头下一步的动作。同时，改变状态寄存器的值，令机器进入一个新的状态。
- 一个状态寄存器。它用来保存图灵机当前所处的状态。图灵机的所有可能状态的数目是有限的，并且有一个特殊的状态，称为停机状态。

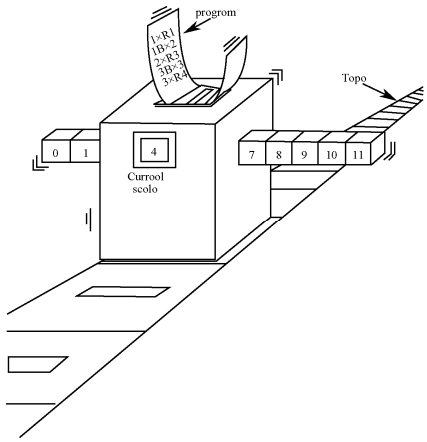


图 1-1 图灵机结构模型

图灵机根据程序的命令和内部的状态在纸带上进行移动和读/写，它的每一部分都是有限的，但它有一个潜在的无限长的纸带。因此，这种机器只是一个理想的设备。图灵认为

这样的一台机器就能模拟人类所能进行的任何计算过程。

2．图灵机的工作过程

将图灵机模型画成二维平面图如图 1-2 所示。首先将输入符号串（有穷长度的从输入字母表中选择的符号串）从左到右依次填在纸带的格子（带单元）上（图 1-2 中用符号 Xi 表示），其他带单元保持空白（即填以空白符，用符号 B 表示）。空格是带符号，但不是输入符号，除了输入符号和空格之外，还可能有其他的带符号。若读/写头（带头）位于某个带单元之上，说明图灵机正在读/写这个单元。

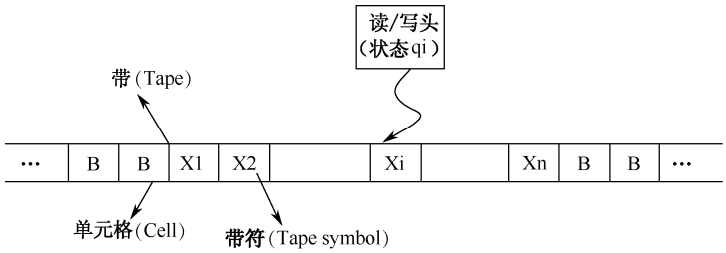


图 1-2　图灵机模型示意图

图灵机的工作过程就是根据读/写头内部程序的命令及内部状态进行纸带的读/写和移动。在初始状态下，读/写头位于输入的最左边单元上（第 0 号格子）。每个图灵机都有一组变换规则，图灵机的移动是当前状态和扫描的带符号的函数，根据当前读/写头指向的带符号及其自身的状态，由规则确定读/写头是否移动及移动方向。每移动一步，图灵机将：

- 改变状态。下一状态可以是任何状态或与当前状态相同。
- 在扫描的单元中写带符号。这个带符号代替扫描的单元中任何符号。所写符号可以是任意的带符号或与当前单元的符号相同。
- 向左或向右移动带头。在本书中要求带头移动而不允许带头保持静止。这个限制并不约束图灵机的计算能力，因为包含静止带头的任何操作都可连同下一个带头移动一起被压缩成单个状态改变、写入新的带符号及向左或向右移动的操作。

例 1-1　假设图灵机纸带上的每个方格内为 a 或 b 或空格 B，控制规则 Table 用于表示当读入（输入）给定时（如读入 a）读/写头的移动方向（向前、向后或停止）。

将输入信息集合表示为 IN，输出信息集合表示为 OUT，且有：

```
IN={x,y,B}
OUT={Left,Right,Stop}
```

设图灵机纸带状态如图 1-3 所示。

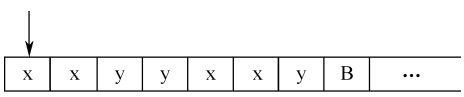


图 1-3　程序示例

给定控制规则 Table 如下：

① 从最左端起始；
② 若读入 x，右移一格；
③ 若读入 y，将其改为 x，再继续右移一格；
④ 若读入 B，停止。

依据上述规则，移动 7 次后，将纸带上停止符 B 左端的符号均改为 x，并停止。

因此，图灵机的工作过程可以简单地描述为读/写头从纸带上读出一个方格中的信息，然后根据它内部的状态对程序进行查表（规则表 Table），得出一个输出动作，确定是向纸

带上写信息还是使读/写头向前或向后移动到下一个方格。同时，程序还会说明下一时刻内部状态转移到哪里。

图灵机只要根据每一时刻读/写头读到的信息和当前的内部状态，查表就可确定它下一时刻的内部状态和输出动作。只要改变规则 Table，图灵机就可以做不同的工作。如果把图灵机的内部状态解释为指令，将规则解释为程序，都用二进制编码表示，与输出和输入信息（也可用二进制码表示）同样存储在机器里，编写不同的程序就会使机器做不同的动作。这就成为电子计算机了。所以，图灵机就是一个最简单的计算机模型。

3．图灵机的形式化描述

形式上，图灵机（TM）可以描述为一个七元组，即

$$M=(Q，\Sigma，\Gamma，\delta，q0，B，F) \qquad (1\text{-}1)$$

其中：

Q：图灵机状态的有穷集合。

Σ：输入符号的有穷集合，不包含空白符。

Γ：带符号的完整集合；Σ 是 Γ 的子集，有 Σ∈Γ。

δ：转移函数。Δ(q，X)的参数是状态 q 和带符号 X。δ(q，X)的值在有定义时是三元组(p，Y，D)，其中：p 是下一状态，属于集合 Q；Y 是在当前扫描的单元中写下的符号，属于 Γ 集合，代替原来单元里的符号；D 是方向，非 L 即 R，分别表示“向左”和“向右”，说明带头移动方向。

转移函数 δ 是一个部分函数，换句话说，对于某些 q 和 X，δ(q，X)可能没有定义，如果在运行中遇到下一个操作没有定义的情况，机器将立刻停机。

q0：初始状态，属于 Q，开始时图灵机就处于 q0 状态。

B：空格符号。这个符号属于 Γ 但不属于 Σ，即不是输入符号。开始时，空格出现在除包含输入符号的有穷多个初始单元之外的所有单元中。

F：终结状态或接受状态的集合，是 Q 的子集。

今天，图灵机依然是计算理论研究的中心课题，图灵机的“停机问题”[1]更是研究许多判定问题的基础。“判定问题”是指判定无穷多个同类个别问题是否具有算法解，或者是否存在能行性的方法使得对该问题类的每一个特例都能在有限步骤内机械地判定它是否具有某种性质的问题。例如，“任给一个素数 x，x 是素数吗？”，这一问题就是判定问题。研究判定问题的意义：如果某问题不可判定，就意味着它是不可解的或不可计算的；相应的，若它是可计算的，则它是可判定的或可解的。由此，又归结到 1.1.2 节结尾处所述的研究可计算性理论的意义。

*1.2 计算机的发展历程

1.2.1 电子计算机的诞生和发展

在 1946 年之前，计算机的工作都是基于机械运行方式，没有进入逻辑运算领域。如果

[1] 停机问题是指是否存在一个算法，对于任意给定的图灵机都能判定任意的输入是否会导致停机？已证明图灵机的停机问题是不可判定的。停机问题的不可判定性成为了解决许多不可判定性问题的基础。

不是 1906 年美国人 Lee De Forest 发明了电子管，电子计算机是不可能出现的。正是电子技术的飞速发展，使计算机由机械式发展到电子时代。

计算机的发展至今经历了五个时代。第一代（1946 年—1954 年）称为“电子管计算机”时代，内部元件使用电子管。图 1-4 所示是第一台用电子管和继电器制作的通用电子计算机 ENIAC，它于 1946 年 2 月 15 日在美国宾夕法尼亚大学问世，共使用了 18800 个电子管，6000 多个开关和配线盘，重约 30 吨，占地 170 平方米，工作主频为 0.1MHz。

美国数学家冯·诺依曼（J·Von Neumann）提出了解决此问题之道，这就是“程序存储方式”。通俗地讲，就是把原来通过切换开关和改变配线来控制的运算步骤，以程序方式预先存放在计算机中，然后让其自动计算。在以后的日子中，计算机的发展正是沿着“程序存储方式”这一光辉道路前进的。

虽然 ENIAC 在每次进行不同的计算时，都需要切换开关和改变配线，使当时从事计算的科学家看上去更像在干体力活。但无论如何，它的诞生，表示人类从此进入了电子计算机时代。从那一天至今的半个多世纪中，随着电子技术的发展，计算机经历了从电子管到晶体管、集成电路、大规模集成电路，以及超大规模集成电路等几代的发展，无论是在体积上、运行速度上，还是在智能性、可靠性及价格等多方面都有了迅猛的进步，成为 20 世纪发展最快的技术，计算机行业也成为 20 世纪最具活力的行业。

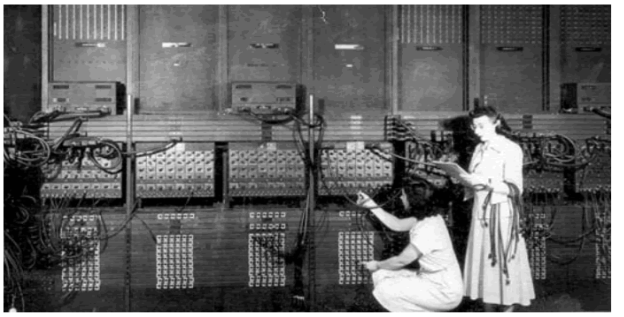


图 1-4　在第一台电子计算机 ENIAC 上编程

第一代计算机主要用于工程计算，其主要特点是采用电子真空管和继电器构成处理器和存储器，用绝缘导线实现互连。体积较为庞大，运算速度较低，运算能力有限。程序编写采用由“0”和“1”组成的二进制码表示的机器语言，只能进行定点数运算。由于电子管易发热，寿命最长只有 3000h。因此，计算机运行时常会因电子管被烧坏而出现死机。

第二代计算机属于晶体管计算机（1960 年—1964 年）。世界上第一台全晶体管计算机 TRADIC 于 1955 年由贝尔实验室研制成功，它装有 800 只晶体管，功率仅为 100W，占地 $3ft^3$，如图 1-5 所示。

第二代计算机采用晶体管逻辑元件及快速磁心存储器，彻底改变了继电器存储器的工作方式和与处理器的连接方法，大大缩小了体积。其运算速度也从第一代的每秒几千次提高到几十万次，主存储器的存储容量从几千字节提高到 10 万字节以上。另外，第二代计算机普遍增加了浮点运算，使数据的绝对值可达到 2 的几十次方或几百次方，同时有了专门用于处理外部数据输入/输出的处理机，使计算能力实现了一次飞跃。除科学计算外，开始被用于企业商务。

在软件方面，第二代计算机除机器语言外，开始采用有编译程序的汇编语言和高级语言，建立了子程序库及批处理监控程序，使程序的设计和编写效率大为提高。

采用集成电路作为逻辑元件是第三代计算机（1964 年—1974 年）的最重要特征。此时，微程序控制、流水线技术、高速缓存和先行处理机等技术开始出现并逐渐普及。第三代计算机的典型代表有 1964 年 IBM 公司研制出的 IBM S/360、CDC 公司的 CDC6600 及 CRAY 公司的巨型计算机 CRAY-1 等，如图 1-6 所示。

图 1-5 TRADIC 晶体管计算机

图 1-6 巨型计算机 CRAY-1

随着集成电路技术的发展，出现了采用大规模和超大规模集成电路及半导体存储器的第四代计算机（1974 年—1991 年），同时，计算机也逐渐开始依据功能和性能的不同分为巨型机、大型机、小型机和微型机，出现了共享存储器、分布存储器及不同结构的并行计算机，并相应产生了用于并行处理和分布处理的软件工具和环境。第四代计算机的代表机型 Cray-2 和 Cray-3 巨型机，因采用并行结构而使运算速度分别达到每秒 12 亿次和每秒 160 亿次。

从 1991 年至今的计算机系统，都可以认为是第五代计算机。超大规模集成电路（VLSI）工艺的日趋完善，使生产更高密度、高速度的处理器和存储器芯片成为可能。这一代计算机的主要特点是大规模并行数据处理、系统结构的可扩展性、高性能的实时通信能力和智能性。随着集成电路技术的不断发展，现代计算机系统的运算速度和整体性能都得到不断提高。图 1-7 所示为中国在 2004 年研制的曙光 4000A 超级计算机，其运算速度达每秒 8.061 万亿次。

图 1-7 曙光 4000A 超级计算机

1.2.2 微型计算机的发展

相对于高性能大型或巨型计算机系统，在 20 世纪 70 年代诞生的微型计算机（又称为 PC，Personal Computer，个人计算机）则因其较高的性价比而在各行各业中得到了更为广泛的应用。

微型计算机的发展伴随的是微处理器的发展。世界上第一片微处理器是 Intel 公司 1971 年研制生产的 Intel 4004（图 1-8），是一个 4 位微处理器，可进行 4 位二进制的并行运算，拥有 45 条指令，速度为 0.05MIPS（Million Instructions Per Second，每秒百万条指令）。

4004 功能有限，主要用于计算器、电动打字机、照相机、台秤、电视机等家用电器上，一般不适用于通用计算机。而在同年末推出的 8 位扩展型微处理器 8008，则是世界上第一片 8 位微处理器，也是真正适用于通用微型计算机的处理器。它可一次处理 8 位二进制数，寻址 16KB 存储空间，拥有 48 条指令系统。这些使它能有机会应用于许多高级的系统。

微处理器及微型计算机从 1971 年至今经历 4 位、8 位、16 位、32 位、64 位及多核芯 6 个时代。除上述主要用于袖珍式计算器的 4004 芯片外，其他具有划时代意义微处理器如下：

- 1973 年，Intel 公司推出的 8 位微处理器 Intel 8080。这是 8 位微处理器的典型代表。它的存储器寻址空间增加到 64KB，并扩充了指令集，指令执行速度达到每秒 50 万条指令，同时它还使处理器外部电路的设计变得更加容易且成本降低。除 Intel 8080 外，同时期推出的还有 Motorola 公司的 MC6800 系列，以及 Zilog 公司的 Z80 等。
- 1978 年，Intel 公司推出的 Intel 8086/8088 微处理器是 16 微处理器的标志。其内部包含 29 000 个 3μm 技术的晶体管，工作频率为 4.77MHz，采用 16 位寄存器和 16 位数据总线，能够寻址 1MB 的内存储器。IBM PC 采用的微处理器就是 8088。同时代的还有 Motorola 公司的 M68000 和 Zilog 公司的 Z8000。
- 1985 年，研制成功的 32 位微处理器 80386 系列。其内部包含 27.5 万个晶体管， 工作频率为 12.5MHz，后逐步提高到 40MHz。可寻址 4GB 内存，并可管理 64TB 的虚拟存储空间。
- “奔腾（Pentium）”微处理器在 2000 年 11 月发布，起步频率为 1.5GHz，随后陆续推出了 1.4～3.2GHz 的 64 位的 P4 处理器。
- 2006 年，开始推出并得到迅速发展的多核处理器，是计算技术的又一次重大飞跃。多核处理器是指在一个处理器上集成两个或以上运算核心，从而提高计算能力。较之单核处理器，多核处理器能带来更高的性能和生产力优势，因而成为了一种广泛普及的计算模式。图 1-9 所示为 Intel 公司推出的双核处理器芯片。

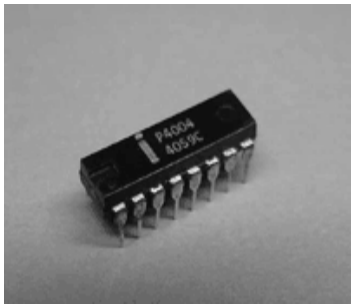


图 1-8　Intel 4004 微处理器芯片

图 1-9　Intel 双核处理器芯片

世界上第一台微型计算机 Altair8800 在 1975 年 4 月由 Altair 的公司推出，它采用 Zilog 公司的 Z80 芯片作为微处理器。它没有显示器和键盘，面板上有指示灯和开关，给人的感觉更像一台仪器箱。

IBM 公司在 1981 年推出了首台个人计算机 IBM PC，1984 年，又推出了更先进的 IBM PC/AT，它支持多任务、多用户，并增加了网络能力，可联网 1000 台 PC。从此，IBM 彻底确立了在微机领域的霸主地位。

今天，微型计算机已真正进入到千家万户、各行各业，它在功能上、运算速度上都已超过了当年的大型机，而价格却只是大型机的几分之一。真正实现了其大众化、平民化和多功能化的设计目标。

1.2.3 未来计算机的发展

未来充满了变数，未来的计算机将会是什么样？

21 世纪是人类走向信息社会的世纪，是网络的时代，是超高速信息公路建设取得实质性进展并进入应用的年代。电子计算机技术正在向巨型化、微型化、网络化和智能化这四个方向发展。

巨型化不是指计算机的体积大，而是指具有运算速度高、存储容量大、功能更完善的计算机系统。巨型机的应用范围如今也日渐广泛，如航空航天、军事工业、气象、电子、人工智能等几十个学科领域发挥着巨大的作用，特别是在复杂的大型科学计算领域，其他的机种难以与之抗衡。

计算机的微型化得益于大规模和超大规模集成电路的飞速发展。现代集成电路技术的发展，已可将计算机中的核心部件：运算器和控制器集成在一块大规模或超大规模集成电路芯片上，作为中央处理单元，称为微处理器，因而才使计算机作为“个人计算机”变得可能。微处理器自 1971 年问世以来，发展非常迅速，伴随着集成电路技术的发展，以微处理器为核心的微型计算机的性能不断跃升。现在，除了放在办公桌上的台式微型机外，还有可随身携带的各种规格的笔记本计算机、可以握在手上的掌上电脑、可随时上网和进行文字处理的平板电脑、手机等。

据美国媒体报道，在 2011 年 2 月，美国科学家已成功研制出世界上最小的计算机——一种可以植入眼球的医用毫米级计算系统。这种计算机主要为青光眼患者研制，放置在患者眼球内可以监测眼压，方便医生及时为病人缓解痛苦。据介绍，这种计算机只有 $1mm^3$ 大小，包括一个极其节能的微处理器、一个压力传感器、一枚记忆卡、一块太阳能电池、一片薄薄的蓄电池和一个无线收发装置。通过无线收发装置，这个计算机能够向外部装置发出眼压数据资料。

从 20 世纪中后期开始，网络技术得到快速发展，已经突破了只是“帮助计算机主机完成与终端通信”这一概念。众多计算机通过相互连接，形成了一个规模庞大、功能多样的网络系统，从而实现信息的相互传递和资源共享。今天，网络技术已经从计算机技术的配角地位上升到与计算机技术紧密结合、不可分割的地位。各种基于网络的计算机技术不断出现和发展，计算机连入网络已经如同电话机连入市内电话交换网一样方便，且网络信息传送的速度也随着“光纤”差不多铺到“家门口”而变得越来越快。今天，计算机技术的发展已离不开网络技术的发展，同时，网络也成为了人类生活的一部分。

计算机的智能化就是要求计算机具有人的智能，即让计算机能够进行图像识别、定理证明、研究学习、探索、联想、启发和理解人的语言等。目前，人工智能技术的研究已取得较大成绩，智能计算机（又称为“机器人”）已部分具有人的能力，能具有简单的“说”、“看”、“听”、“做”能力，能替代人类去做一些体力劳动或从事一些危险的工作，如日本福岛核电站出现核泄漏后，日本政府就曾“派”机器人进入核电站检测核泄漏的情况。

人工智能是目前以至未来可见的时间里计算机科学的研究热点。人工神经网络的研究，

使计算机向人类大脑接近又迈出了重要的一步。今天，除了希望在软件技术上不断深入研究，人们还寄希望于全新的计算机技术能够带动人工智能的发展。至少有三种技术有可能引发全新的革命，它们是光子计算机、生物计算机和量子计算机。

光子计算机的运算速度据推测可能比现行的超级计算机快 1000～10 000 倍。而一台具有 5000 个左右量子位的量子计算机可以在大约 30s 内解决传统超级计算机需要 100 亿年才能解决的素数问题。相对而言，生物计算机研究更加现实，美国威斯康星－麦迪逊大学已研制出一台可进行较复杂运算的 DNA 计算机。据悉，一克 DNA 所能存储的信息量可与 1 万亿张 CD 光盘相当。这些推测，使人们有理由对人工智能的发展前景变得乐观。

计算机真的能达到人类的思维能力，模拟人类的行为动作吗？未来的计算机会像影视剧中描述的那样完全达到人的智力吗？

1.3 计算机中的信息表示

如今，信息是一个非常流行的词汇。人际社会中，每天都少不了信息交互，每个人都是信息的发布者，同时也都是信息的接收者。互联网上，更是每分每秒都有大量的信息在传送。信息交换和信息共享促进了新知识的传播、新价值的产生，也推动着社会的进步。

在这个“信息爆炸”的时代，对信息的传播、处理和存储都离不开计算机这个载体。本节就在给出“信息”一词一般描述的基础上，介绍计算机中的信息表示和基于计算机的信息处理过程。

1.3.1 信息

对“信息”一词最早的解释见于哈特莱（Ralph V.L.Hartley）1928 年发表在《贝尔系统技术杂志》上的“信息传输”一文中，他把信息理解为选择通信符号的方式，并用选择的自由度来计量这种信息量的大小。信息论（Information Theory）的开山鼻祖、美国数学家香农（C.E.Shannon）1948 年在《贝尔系统技术杂志》上发表了一篇题为“通信的数学理论”论文，该文被认为是信息论诞生的标志。香农以概率论为工具，阐述了通信工程中的一系列基本理论问题，建立了信息从信源（发送方）通过信道（传输途径）传递给信宿（接收方）的通信系统模型，并给出了计算信源信息量和信道容量的方法和计算信息熵的公式。他对信息的解释：信息是用来减少随机不定性的东西。控制论创始人之一，美国科学家维纳（N.Wiener）指出：信息就是信息，既不是物质也不是能量，专门指出了信息是区别于物质与能量的第三类资源。

《辞源》中将信息定义为“信息就是收信者事先所不知道的报道”。作为科学术语，可以简单地将信息理解为消息接收者预先不知道的报道”。我国学者钟义信从认识论的层次将信息定义为“主体关于某事物的认识论层次信息，是指主体所感知或表述的关于该事物的运动状态及其变化方式，包括状态及其变化方式的形式、含义和效用”。

对于信息的定义，至今仍是众说纷纭，莫衷一是。但人们对信息的共同认识：信息是一种宝贵的资源，信息、材料（物质）、能源（能量）是组成社会物质文明的三大要素。

相对于通信范围内的信息论（狭义信息论），广义信息论以各种系统、各门科学中的信息为对象，以信息过程的运动规律作为主要研究内容，广泛研究信息的本质和特点，以及

信息的取得、计量、传输、储存、处理、控制和利用的一般规律，使得人类对信息现象的认识与揭示不断丰富和完善。所以，广义信息论又称为信息科学，它以信息为主要研究对象，是一门新兴的跨跃多个学科的科学。

在一般用语中，信息、数据、信号并不被严格区别，但从信息科学的角度看，它们是不能等同的。在用现代科技（计算机技术、电子技术等）采集、处理信息时，必须要将现实生活中的各类信息转换成智能机器能识别的符号（符号具体化即是数据，或者说信息的符号化就是数据），再加工处理成新的信息。数据可以是文字、数字或图像，是信息的具体表示形式，是信息的载体。而信号则是数据的电磁或光脉冲编码，是各种实际通信系统中，适合信道传输的物理量。信号可以分为模拟信号（随时间而连续变化的信号）和数字信号（在时间上的一种离散信号）。

1.3.2 数值信息表示

现代计算机中所有的数都采用二进制，无论数的大小，都是 1 和 0 的组合。每位 1 或 0 称为 1 位二进制码。例如，8 需要用 4 位二进制码表示（1000），100 要用 8 位二进制码表示（1100100），而要表示 1 万，则至少需要用 14 位二进制码（10011100010000）。由此可见，同样大小的数，二进制要比对应的十进制数书写长度要长很多。

也许是人有 10 个手指的缘故，人类从结绳计数开始，就一直采用十进制。也就是说，我们已经很习惯于用十进制来表达数的大小及对数据的运算。那么，为什么人类发明的计算机却要采用二进制呢？

香农在他的“通信的数学理论”论文中曾首次指出，通信的基本信息单元是符号（Smmble），而最基本的符号是二值符号，又称为二进制码。

图 1-10 太极八卦图

二进制的灵感据说来自于中国的太极八卦图，如图 1-10 所示。1701 年，德国数学家莱布尼茨（G • W • Leibniz）受中国八卦图的启发[1]，发明了二进制。

太极（伏羲）八卦图分为“阴”、“阳”两仪组成。两仪生四象：太阴（00）、少阳（01）、少阴（10）、太阳（11）；四象生八卦：乾（111）、兑（110）、离（101）、震（100）、巽（011）、坎（010）、艮（001）、坤（000）。由此，通过阴阳引申，就可以表示宇宙万物。如果将“阴”视为 0，将“阳”视为 1，所有的卦象于是也就可以看成 0 和 1 的组合。例如：

太阴——00，少阳——01，少阴——10，太阳——11；

乾——111，兑——110，离——101，震——100，巽——011，坎——010，艮——001，坤——000。

这样，用 6 位 0 和 1，就可以表示八卦图的 64 个卦象。

莱布尼茨的二进制，就是用 0 和 1 去表示一切数字，如 000，001，010，011 就分别代表 0～3 这 4 个数字。1848 年，英国数学家乔治 • 布尔（George Boole）推出了二进制运算

[1] 胡阳、李长锋在《莱布尼茨－二进制与伏羲八卦图考》一书中论证了莱布尼茨的二进制至少在某种程度上受到了八卦图的启发。

法则，为二进制计算机的诞生奠定了基础。

诞生于 1946 年的第一台电子计算机 ENIAC 采用的是十进制，可以同时处理 10 个十进制数。由于十进制有 10 个符号，意味着需要有 10 种稳定状态与之对应，不仅造成数据量大、工作速度低，更主要是用电子器件实现起来很困难。因此，冯·诺依曼[1]提出了在计算机中采用二进制。采用二进制主要有以下理由：

（1）二进制只有 0 和 1 两个基本符号，任何两种对立的物理状态都可以归结为用二进制表示。例如，开关的“闭合”与“断开；电位的“高”和“低”；晶体管的“导通”与“截止”；电容的“满电荷”与“空电荷”等。如此，一切有两种对立稳定状态的器件都可以表示二进制的“0”和“1”。

我们可以用一个简单的示例来说明。在图 1-11 中，当图 1-11（a）中的 X 端电位为 0V 时，晶体二极管导通，有电流流过电阻 R；当 X 端电位为+5V 时二极管将截止，R 上将不会有电流流过。根据欧姆定理知，导通时 a 点电位≈0V（低电平）；截止时因电流 i=0，则 a 点电位=+5V（高电平）。如果周期性地使 X 端呈现 0V 和+5V（如图 1-11（b）所示的脉冲波），则二极管就会周期地导通和截止。如果将 0V 用“0”表示，5V 用“1”表示，则上述过程就与二进制编码对应了。

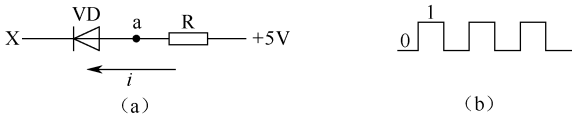


图 1-11　二极管电路与二进制编码

具有两种稳定状态的电子元件很容易找到，产生两种稳定状态的电路也易于设计。因此，计算机采用二进制的重要原因之一就是其非常容易用电子器件实现，可靠性也高。

（2）二进制的另外一个主要优点是算术运算规则简单，且适合逻辑运算。二进制数的算术运算特别简单，加法和乘法各仅有 3 条运算规则（加法：0+0=0，0+1=1，1+1=10；乘法：0×0=0，0×1=0，1×1=1）。二减法和除法则可以通过一定的变换转换为加法和乘法运算[2]。

计算机的基本运算是算术运算和逻辑运算。逻辑运算[3]的对象是“真”和“假”，二进制数的“1”和“0”正好可与逻辑值“真”和“假”相对应，这就使计算机进行逻辑运算变得非常方便。图 1-12 分别给出了用二极管实现的“与”逻辑和“或”逻辑运算的电路示例。

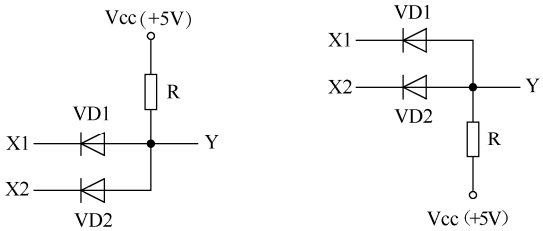


图 1-12　二极管逻辑门电路

[1] 有关冯·诺依曼的介绍见第 3 章。
[2] 将减法运算转换为加法运算的原理见 2.3 节（二进制的表示和运算）。除法到乘法的转换请参阅有关计算机原理方面的书籍。
[3] 关于逻辑运算的详细介绍见 2.4 节（逻辑运算与逻辑门）。

计算机可以说是由许许多多个类似图 1-12 这样的门电路组成的，它用“1”表示高电平，用“0”表示低电平。这样，就将电位状态与二进制编码一一对应起来。因此，二进制是计算机硬件唯一能够直接识别的编码。

1.3.3 文字信息表示

由上述分析已知，计算机能够直接识别的只有二进制码。所以，要让计算机保存或处理的所有信息都必须采用二进制码表示，文字信息也不例外。

文字由字符组成。计算机是美国人发明的，所以，计算机中的文字首先是西文字符，包括字母、数字、符号及特殊控制字符。西文字符编码方式很多，目前，国际上广泛使用的是 ASCII 码（American Standard Code for Information Interchange，美国标准信息交换码），分为标准 ASCII 码和扩展 ASCII 码两种。

标准 ASCII 码的用 7 位（bit）二进制码（bit_6～bit_0）表示，总共可表示 128 个字符（知道为什么是 128 个吗？）。但由于计算机从诞生那天起，能够同时处理（专业上称为并行处理）的二进制数就是 8 位（从来没有 7 位）。因此，标准 ASCII 码实际上是用 8 位二进制码来表示的，8 位二进制码又称为 1B（Byte），在内存中占用 1 个单元（这些名词很陌生，没关系，先留着，后边就学到了）。最高位（bit_7）用作奇偶校验位（默认情况下为 0）。奇偶校验，是指在代码传送过程中用来检验是否出现错误的一种方法，分为奇校验和偶校验两种。奇校验规定：正确的代码一个字节中 1 的个数必须是奇数，若非奇数，则使最高位 bit_7 为 1（补为奇数）；偶校验规定：正确的代码一个字节中 1 的个数必须是偶数，若非偶数，则使最高位 bit_7 为 1。标准 ASCII 码表示的 128 个字符（见附录 B）中，包含 10 个阿拉伯数字，52 个英文大小写字母，33 个符号及 33 个控制符。一个字符对应一个编码。如字符“A”对应的 ASCII 码为 65，而空格对应的 ASCII 码为 32。

为了表示更多的欧洲常用字符（如德语中的字母 ü），对标准 ASCII 进行了扩展。扩展 ASCII 由 8 位二进制数码组成，这样就可以表示 256 种不同的符号。

除 ASCII 码外，较常见的西文字符编码还有 EBCDIC 码，用 8 位二进制码表示，可表示 256 个字符。

为了使普通国人也能使用计算机，需要计算机能够处理汉字。数值和西文字符可以通过键盘直接输入（谁让计算机是人家发明的呢？），而汉字是象形文字，计算机处理汉字的关键首先是如何将每个汉字变成可以直接从键盘输入的代码——即汉字的外码，然后再将输入码转换为汉字机内码，之后才能对其处理和存储。在输出汉字时，则需进行相反的过程，将机内码转换为汉字的字型码。因此，汉字的编码包括外码、机内码、字形码和矢量汉字。

汉字的外码即它的输入码，目前，常见的编码法有拼音、五笔和搜狗等。

机内码主要有国标码、BIG5 码（主要在中国台湾和香港地区使用）等。我国国家标准局于 1981 年颁布了“国家标准信息交换用汉字编码基本字符集”（GB2312—80），共收集了 6763 个汉字，682 个非汉字符号（外文、字母、数字和各种图形等），每个汉字对应一个国标码。

每个国标码用 2B 表示，为避免与 ASCII 码冲突，规定汉字国标码每个字节的最高位为“1”，即首位是“0”为字符，首位是“1”为汉字。这样的“国标码”就是汉字在计算机中的表示，也就是机内码。

另一种可以在计算机中表示汉字的编码为 Unicode 编码（Universal Multiple Octet Coded Character Set）。Unicode 是国际标准组织针对各国文字和符号进行的、在计算机上使用的统一性字符编码，它为每种语言中的每个字符设定了唯一的二进制编码，以满足跨语言、跨平台进行文本转换、处理的要求。

字形码是确定一个汉字字形点阵的代码，字形点阵中的每个点对应一个二进制位。每个汉字对应一个点阵，再编上代号存入存储器中，这就是字模库。汉字在显示时需要在汉字库中查找汉字字模并以字模点阵码形式输出。

汉字的另一种显示方式是矢量汉字。矢量字库保存每一个汉字的描述信息，如一个笔划的起始、终止坐标，半径，弧度等。在显示、打印这一类字库时，需经过一系列的数学运算才能输出结果。

点阵字库的汉字由若干个点组成，当字体放大时，点会随之放大，使得字看上去比较“粗”。矢量字库保存的汉字理论上可以被无限放大，笔划轮廓仍然能保持圆滑清晰。打印时使用的字库均为矢量字库。Windows 使用的字库为以上两类，在操作系统的“WINDOWS\Fonts”目录下，如果字体文件后的扩展名为 FON，表示该文件为点阵字库；若扩展名为 TTF，则表示是矢量字库。

汉字信息从输入到输出的处理过程如图 1-13 所示。

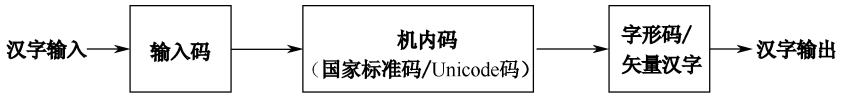


图 1-13　汉字代码的处理过程

1.3.4　声音与图像信息表示

计算机中存储和处理的信息除数值和文字外，还有各类被称为多媒体的信息，包括声音、图像和视频等。与数值和字符信息不同，这些信息都是连续变化的模拟信号，无法直接用计算机进行存储和处理，必须要首先转换为由 0 和 1 组成的二进制位串，这一过程称为数字化。

1．声音信息的表示

声音是通过空气传播的一种连续的波（Sound Wave，声波），它的连续性体现为幅值大小是连续的，可以是实数范围内的任意值；在时间上是连续的，没有间断点。我们将这种在时间和幅值上都连续变化的信号称为模拟信号。相应的，将时间和幅值都不连续的信号称为离散信号，如图 1-14 所示。

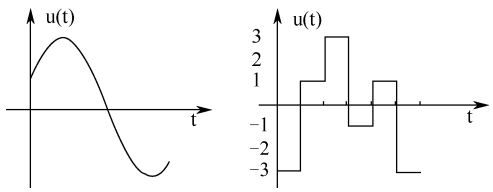


图 1-14　模拟声音信号和数字声音信号

要使声音信号能够被计算机处理，首先需要对起进行数字化，即将时间和幅值均连续变化

的模拟声音信号，通过采样（Sampling）和量化（Measuring），转换为时间和幅值均不连续的离散信号，这种离散的声音信号称为数字音频信号，也就是计算机能够存储和处理的信号。

数字声音在计算机存储器中的存放形式称为声音文件格式。相同的数据，可以有不同的存放形式，所以，也就有多种文件格式，不同的格式其文件的扩展名不同（如*.WAV），每种格式都具有特定的应用场合。计算机中广泛应用的数字化声音文件有两类，一类是采集各种声音的机械振动得到的数字文件（又称为波形文件），其中包括音乐、语音及自然界的效果音等，另一类是专门用于记录数字化乐声的 MIDI 格式文件。常见的波形声音文件格式有 WAV 文件、MP3、RealAudio 等。

- WAV 格式是微软公司开发的一种声音文件格式，又称为波形声音文件，是最早的数字音频格式，被 Windows 平台及其应用程序广泛支持。主要用于自然声的保存与回放，其特点是声音层次丰富，还原性好，表现力强。如果使用足够高的采样频率和采样精度，可以获得极好的音质，但文件的数据量比较大。该格式的文件可以被几乎所有的多媒体软件使用，易于编辑。
- CD 存储格式与 WAV 一样，采样频率为 44.1kHz，量化位数 16 位，记录的是波形流，是一种近似无损的格式。
- MP3 格式（全称为 MPEG，Moving PictureExpert Group-1 Audio Layer 3）属于压缩存储格式，它能够在音质丢失很小的情况下（人耳根本无法察觉这种音质损失）把文件压缩到更小的程度，从而大幅降低存储的数据量。
- RealAudio 是由 Real Networks 公司推出的一种文件格式，最大的特点就是可以实时传输音频信息，尤其是在网速较慢的情况下，仍然可以较为流畅地传送数据，主要适用于网络上的在线播放。现在的 RealAudio 文件格式主要有 RA（RealAudio）、RM（RealMedia，RealAudio G2）和 RMX（RealAudio Secured）三种。
- QuickTime 是苹果公司于 1991 年推出的一种数字流媒体，它面向视频编辑、Web 网站创建和媒体技术平台，QuickTime 支持几乎所有主流的个人计算平台，可以通过互联网提供实时的数字化信息流、工作流与文件回放功能。

MIDI（Musical Instrument Digital Interface，乐器数字接口）是数字音乐/电子合成乐器的统一国际标准。它定义了计算机音乐程序、数字合成器及其他电子设备交换音乐信号的方式，规定了不同厂家的电子乐器与计算机连接的电缆和硬件及设备间数据传输的协议，可以模拟多种乐器的声音。凭借各种 MIDI 软件工具、个人计算机和 MIDI 硬件，作曲家可以作出复杂的、具有专业水平的乐曲。

MIDI 文件就是 MIDI 格式的文件，以.mid、.cmf 或 .rol 文件扩展名进行存储。在 MIDI 文件中存储的是一些指令。把这些指令发送给声卡，由声卡按照指令将声音合成出来。MIDI 文件比波形文件更为紧凑。3min 的 MIDI 音乐仅仅需要 10KB 的存储空间，而 3min 的波形音乐则需要 15MB 的存储空间。

声音文件的顺利播放，取决于播放器（播放声音的软件）能否正确识别相应的文件格式。一种声音文件可以由一种以上的播放器播放，一种播放器也能播放多种声音文件。

2．图像信息的表示

俗话说“百闻不如一见”，人类从自然界获取的信息中，视觉信息占了极大的比重。

有些花费很多笔墨也很难表达清楚的事物，若用一幅图像描述，可以做到“一目了然”。例如，一本好的家电或设备的使用说明书中，总是在文字说明的同时配有详细地操作示意图，阅读这些简图就比较容易理解相应的文字说明，并大致了解设备的基本构造和使用方法。因此，图像也是计算机处理的重要信息类型。

“图”，包括图像和图形。图像（Image）是自然界的景物通过人们的视觉器官在大脑中留下印象。我们常见的各种照片、图片、海报、广告画等均属于图像。图像可以是简单的黑白图像，也可以是全真色彩的照片。最简单的图像是单色图像（二值图像），所包含的颜色仅有黑色和白色两种。彩色图像包含了各种色彩（颜色）。

图形（Graphics）又称为矢量图，是通过数学公式计算、由程序设计语言实现的图，使用直线和曲线来描述。例如，对于直线，可以通过 line，start_point，end_point 表示；对于圆，则表示为 circle，center_x，center_y，radius；而一幅花的矢量图可以由线段形成外框轮廓，通过设定外框的颜色及外框所封闭的颜色决定花显示出的颜色。

日常生活中看到的图像都是色彩（或灰度）连续变化的模拟图像，如用胶卷拍出的相片就是模拟图像，它的特点是空间上是连续的，可以洗一寸的照片也可以洗二寸的照片，不影响视觉效果。与声音信号一样，要使图像能为计算机处理和存储，必须将其离散化，即转换为数字图像。

图像是在二维空间坐标上连续变化的函数，连续图像的数字化过程是空间和幅值的离散化。例如，将图 1-15 所示的二值图像离散化后（用 0 表示黑色，用 1 表示白色），就转换为图 1-16 所示的计算机能够存储和处理的数字图像。

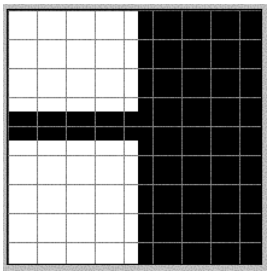


图 1-15　二值图像

1	1	1	1	1	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0
1	1	1	1	1	0	0	0	0	0

图 1-16　图 1-15 在计算机中的表示

图像数据在计算机中的存放形式称为图像文件格式。常用的图像文件格式有位图文件格式（BMP）、索引文件格式（GIF）和 JPEG 压缩文件格式（JPG）等。

BMP 格式是 Windows 采用的图像文件存储格式，在 Windows 环境下运行的所有图像处理软件都支持这种格式，能够在任何类型的显示设备上显示。BMP 位图文件默认的文件扩展名是 BMP 或 bmp。位图文件是一种不压缩的存储格式，图像质量较高，没有数据损失，但占有的存储空间较大。

GIF（Graphics Interchange Format）格式是 CompuServe 公司开发的图像文件格式，属于压缩存储方式，因此，占用的存储空间很小，在网络中被广泛采用。GIF 格式文件还支持透明图像属性和动画图像属性，但表示的颜色数量有限，适合存储颜色较少的卡通图像、徽标等手绘图像。

JPEG（Joint Photographic Experts Group）是由 ISO（International Standard Organization，国际标准化组织）和 IEC（International Electrotechnics Committee，国际电工委员会）两个

组织联合开发的一种算法，称为 JPEG 算法，又称为 JPEG 标准，相应的文件存储格式为 JPG（或 jpg）格式。JPEG 是一个适用范围很广的静态图像数据压缩标准，既可用于灰度图像又可用于彩色图像。

JPG 文件在压缩时可以在调节图像的压缩比和图像保真度，从而根据需要得到不同质量和不同文件大小的图像。JPG 格式的文件比较适合存储色彩丰富的照片，虽然数据压缩是图像数据有所损失，但在一定分辨率下，视觉感受并不明显，因此，得到了软件、硬件厂商的普遍支持，几乎所有数字照相机中存放的都是 JPG 格式的照片文件。

有关数字化声音和数字化图像的进一步描述见附录 C。

为了提高数字化后的声音和图像质量，经采样、量化后的数字音频信号和数字图像通常还需要经过一定的处理，如去噪、锐化等。对音频信号，可以添加各种效果（如淡入淡出、频率均衡和混响等）；对图像信号，若需进一步分析，还需要特征提取、图像分割等各种图像处理技术。对此有兴趣的读者可参阅关于图像处理的相关书籍。

*1.3.5 计算机中信息处理的一般过程

在信息时代，要使各种信息能够最大限度地发挥作用、为人类所用，必须利用计算机这个信息时代最为重要的工具。只有通过计算机高效处理，信息才能真正在广泛的领域中实现“可用”。例如，只有通过计算机对大量气象数据信息进行快速分析和计算，才能有准确的天气预报；各种商务网站因为有计算机的管理，才有可能为用户提供网上购物的服务等。

利用计算机实现对信息的处理和利用，需要经过以下过程，即信息采集、信息表示和压缩、信息存储和组织、信息传输及信息检索。

1．信息采集

要利用计算机对信息进行处理，需将现实生活中的各类信息转换成智能机器能识别的符号，然后才能再加工处理成新的信息。将信息转换成具体的符号就是数据，可以说数据是信息的符号化，是信息的具体表示形式。数据可以是文字、数值、声音、图像和视频等。

对文字和数值信息的采集方法很多，传统的有键盘输入，随着技术的发展，现在语音输入、手写输入、扫描加模式识别等输入方法也日渐普及。

声音、图像和视频信息常简称多媒体信息。对这类信息的采集方法很多，常用的如录音笔、数码照相机、数码摄像机等。

需要明确的一点是，不论是哪种设备和方法，它们所采集的各类信息最终在计算机中都必须转换为计算机能够识别和处理的、由“0”和“1”组成的二进制码，而这些二进制码都可以称为数据。

我们已经知道，二进制的“0”和“1”，在计算机中实际上是“低电平”和“高电平”。所以，一串二进制码数据事实上是一串电脉冲信号，或者说信号是数据的电磁或光脉冲编码，是各种实际通信系统中，适合信道传输的物理量。

“信息”、“数据”和“信号”这三个名词，严格地讲，是三个不同的概念。但在基于计算机的信息处理中，它们相互间是有关系的。信息的采集需要通过信道以信号的形式输入到系统，再转换为具体的符号，也就是数据，进行处理。

2．信息表示和压缩

计算机可处理的信息包括数值、文字、声音、图形图像和视频。1.3 节中对除视频之外的各类信息的表示都已做了简要的描述。

信息采集到计算机中后需要存储。虽然现代计算机中存储设备的容量在不断增大，但受信道传输效率、硬盘容量、处理器速度等各种因素的限制，在很多情况下，仍然希望能够在尽可能小的数据量中包含尽可能多的信息。

通常，多媒体信息的数据量都较大。例如，一张写满文字的 A4 纸的数据量约为 50KB，一幅 3264×2448 的未经压缩的照片的数据量约为 4MB，1min 的 MPEG1 压缩视频大约需要 10MB 的存储空间。庞大数据量造成传输和存储的不便，因此，需要对采集的信息进行压缩。

压缩的任务就是在保持信源信号在一个可以接收质量的前提下，把需要的数据量（比特数）减到最少程度，以减少存储和传输的成本。

有关数据压缩的详细介绍请参阅相关专业书籍。

3．信息存储和组织

今天的时代称为信息爆炸时代，通过网络等各种媒介可以获得大量的各类信息。那么，我们是否考虑过这些信息、特别是电子信息是如何存放的？为什么可以快速地找到它们？这就是信息的组织。

计算机通过文件和数据库技术来对信息进行组织和管理。文件是指存放于计算机中、具有唯一文件名的一组相关信息的集合。计算机中所有的信息包括各种不同类型的程序都是以文件的形式存放的。文件包括有结构文件和无结构的流式文件两种类型。流式文件指由字符序列集合组成的文件。例如，一个源程序文件。

有结构文件（图 1-17）由一条条记录组成，如一件商品的名称、规格、生产厂家、价格等，就可以形成一条记录。

商品编号	商品名	规格	定价	生产厂
100001	圆头螺钉	M4	¥1.20	西安金属材料厂
100002	方头螺钉	M6	¥2.50	西安金属材料厂
100003	六角螺钉	M6	¥3.00	西安金属材料厂
200001	螺母	M4	¥0.60	西安金属材料厂
200002	螺母	M6	¥0.80	西安金属材料厂
200003	六角螺母	M6	¥1.80	西安金属材料厂

图 1-17　有结构文件示意图

一组同类的记录可以形成一个文件；一组相关的文件可以形成数据库。

文件系统使用方便，但存在很多缺陷，如数据冗余（同样的数据出现在多个文件中），数据不一致性（当一个文件中数据被修改时，另一个文件中的相同信息没有同步修改），安全性差等。

为了弥补传统文件系统的不足，诞生了数据库（Database，DB）技术。就像仓库是用于存放产品一样，数据库中存放的是就是大量的数据。仓库中存放的产品会按类型、用途、生产厂商等分门别类排放在不同的地方，并且由统一的仓库管理员进行相应的入库和出库登记。数据库中的数据也按一定的规则存放和管理，并为不同的用户提供需要的数据服务

和信息共享。对应于仓库管理员，数据库的管理由数据库管理系统（Database Management System，DBMS）完成。它是一种软件产品，对数据库中的数据进行集中有效的管理，并为应用程序提供数据资源访问。事实上，我们之所以能够通过网络查询到各种需要的信息，除搜索引擎（如“百度”）外，还要依赖于各个 DBMS 从数据库中去找到相应的数据。

数据库与文件系统最主要的不同是，数据库中的数据是相关的。例如，一位在校学生可能有个人学籍信息、健康信息、选课信息等，这些信息会出现于不同的表格存放在不同的部门，当某学生中途离校，该生的学籍信息会取消，同时其他所有表格中该生的信息也都会被修改。这一点是文件系统所无法实现的。当然，除此之外数据库技术还克服了上述文件系统所故有的其他缺点。图 1-18 是一个产品管理数据库系统示意图。

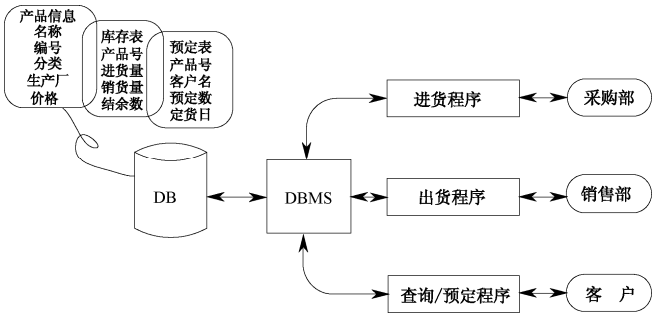


图 1-18 产品管理数据库系统示意图

4．信息传输

信息传输需要通过互联网。今天，可以通过电子邮件、即时通信等方法与世界各地的朋友取得联系；为了获取知识，可以利用搜索引擎将存放在世界各地不同数据库中的信息反馈到查询者的计算机屏幕上。这些，都需要基于网络技术。

计算机网络源于计算机与通信技术的结合，它利用各种通信手段，例如，电话线、同轴电缆、无线电线路、卫星线路、微波中继线路、光纤等，将地理上分散的计算机有机地连在一起，相互通信而且共享软件、硬件和数据等资源。网络技术始于 20 世纪 50 年代，自诞生至今发展迅猛，由主机与终端之间的远程通信，发展到今天世界上成千上万台计算机的互联，形成了遍布全球的互联网（Internet），从而使网络成为生活中不可或缺的一部分。

有关计算机网络的基础知识将在第 4 章中介绍。

5．信息检索

当在科学研究中需要查阅相关文献时，当我们希望到网上商城去“淘”一点价廉物美的商品时，搜索引擎为我们提供了强大的信息检索功能，实现从海量数据中查找到所需要的信息。这就涉及到信息检索技术。

信息检索技术是在计算机技术和网络技术的基础上发展和完善起来的，在经历了手工检索、计算机脱机检索、联机检索阶段之后，如今已实现了网络化，并向更高级的智能化发展。

信息检索是指将杂乱无序的信息有序化后形成信息集合，并根据需要从信息集合中查找出特定信息的过程。实现检索的前提（或基础）是要将一定范围内的信息进行筛选、特

征描述、有序化处理，形成信息集合，即建立数据库。而检索则是采用一定的方法与策略从数据库中查找出所需信息。所以，信息检索也可以简单地理解为信息查找（Information Search）。

信息检索的实质是将用户的检索标识与信息集合中存储的信息标识进行比较与选择或称为匹配（Matching），当用户的检索标识与信息存储标识匹配时，信息就会被查找出来，否则就查不出来。匹配有多种形式，既可以是完全匹配，也可以是部分匹配，这主要取决于用户的需要。

在以上讨论的基于计算机的信息处理中，计算机的硬件和操作系统是平台，网络是信息传输和检索的通道，信息组织、管理及信息处理等都需要利用计算机程序设计语言去实现。

1.4 基于计算机的问题求解

计算机学科要解决的根本问题，就是“利用计算机进行问题求解”。因此，有必要首先了解一下利用计算机进行问题求解的一般过程。

日常生活中，每个人每天都会碰到大大小小的各种问题。碰到问题就需要解决问题。那么，我们是否总结过在遇到问题时是怎样建立起解决问题的思路，又是怎样选择了解决问题的方法呢？

事实上，每个人在遇到问题时，都有意或无意识地经历了以下这样的过程：

- 首先，都会下意识地先对问题是否可解决做一个快速评估，即该问题是否可解决。
- 会确定解决问题的方法。对一个简单的问题，可能不需要过多地思考就能立刻解决。但如果是个大问题（系统性问题），可能就需要对问题进行分解，分配给多个人、花费一定的时间去完成。
- 每个领到任务的人，会确定解决这个具体问题的方法并完成它。
- 当每个人分配到的“子问题”都解决之后，需要将所有的结果汇总在一起，以构成对大问题的解决结果。为了保证能够将“答案”合成到一起，需要在分配任务时就要说清楚提交“答案”的格式。
- 最后，我们需要确认问题解决的正确性。

这是人类对大问题求解的过程，也是计算机求解系统性问题的一般过程。假设某大学要开发一套利用计算机进行控制的课程管理系统。对这个问题，需要做哪些工作呢？

作为计算机专业人员，可能对学校的学生管理模式不了解。因此，首先需要了解这个系统应该有些什么功能，这需要和学校有关人员进行沟通。但学校的管理者对计算机技术可能不熟悉，他们不清楚哪些是计算机可以或容易实现的，而哪些是很难以实现的（计算机不是万能的）。因此，这样的沟通可能需要若干次，沟通的最终目的是要弄清楚用户的需求。所以，这个过程被称为“需求分析”。

在需求弄清楚之后，就进入了系统的设计阶段。它包括对功能模块的划分、算法设计、程序编写和调试，以及最后的系统测试。或许在设计甚至程序编码阶段，会突然发现当初对用户的需求没有理解对，或用户需求又发生了变化，那么就需要从新在修改需求，并依次修改后续的设计、编码等。

理论上，上述这个过程中的一个步骤结束后才可以开始下一个步骤，但它们又是可以反馈的。下一个步骤进行中可能会发现上一步存在某些不足或不完善处，此时就需要返回去修改。对系统性问题的求解过程可以用如图 1-19 所示的模型表示。

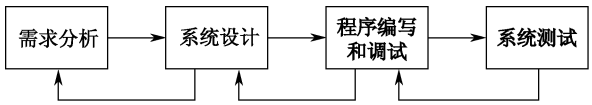


图 1-19 系统性问题求解的一般过程

1.4.1 需求分析与模型建立

需求分析的主要任务：在充分理解了用户需求（如课程管理系统需要有哪些功能）及对目标系统（如课程管理系统）领域知识有一定了解的基础上，确定问题“是否可解决”及“解决什么”。

“是否可解决”即解决问题的可行性。包括计算机科学求解问题的局限性、现有人员的技术水平、可能存在的经济和技术风险等；而“解决什么”则是在了解了问题所具有的特征、特点等基础上，对问题进行抽象，从而建立起系统模型。

从物理域中弄清楚要解决的问题，并通过对问题的抽象建立起逻辑模型，是一个复杂的过程，主要存在以下几个难点：

- **问题的复杂性**。用户需求涉及的因素繁多，如运行环境和系统功能等，引起问题的复杂性。
- **交流障碍**。需求分析涉及人员较多，这些人具备不同的背景知识，处于不同角度，扮演不同角色，造成相互之间交流困难。
- **不完备性和不一致性**。用户对问题的陈述往往是不完备的，各方面的需求可能还存在矛盾，需求分析要消除矛盾，形成完备及一致的定义。
- **需求易变性**。把握需求不是一蹴而就的，因此，变化是客观的。

需求分析的主要的工作如下：

- **问题识别**。要确定用户对目标系统的综合要求（这里的目标系统就是按照用户需求、最终由计算机软件实现的系统），提出这些需求实现的条件，以及最后应达到的标准。包括功能、性能、可靠性、用户界面等。
- **分析建模**。通过对问题的分析和方案的综合，逐步细化和明确目标系统的各项功能，建立问题求解的模型。
- **编写需求分析文档**。包括编写“需求规格说明书”、“初步用户使用手册”、“确认测试计划”等。

需要说明的一点是，我们常说的计算机软件并不仅指程序，它还包括在整个问题求解过程中所编写的各种相关文档资料（这可是相当重要的），即计算机软件是程序、数据，以及各种相关文档资料的总和。上述的需求分析文档就是这些文档中的一部分。

总之，需求分析的主要目标就是建立起系统的逻辑模型。模型，是指对某一真实系统（如课程管理系统、学籍管理系统等）的目标、结构、行为等的抽象描述。模型的内容一般包括对象（概念）和对象之间的关系。建模的过程就是识别概念和概念间关系，并利用对象、联系等基本模型元素来描述系统的结构和行为等。

建模的根本是抽象，抽象是将问题域中的各种人、物、人或物之间的联系抽取出来，用一些特有的符号或形式表示。简单地讲就是对欲求解问题给出清晰的定义和描述。例如，对课程管理系统，系统中涉及了哪些对象？它们都有哪些属性？这些对象间有什么样的关系？有什么样的输入和输出？系统需要处理的信息及做什么样的加工等。这些问题看上去很玄，但利用课程管理这个简单系统，读者也许就能初步地理解。

经过对课程管理系统的需求分析，可以确定出系统中应包含的对象有教师、学生和课程。每个对象都有它们的属性，如学生的属性有学号、姓名、性别、年龄、班级等；课程的属性可以有课程代号、课程名、学分、学时等。教师和学生的关系可以是一位教师带多位学生，称为 1 对多关系。系统的输入就是已知什么条件，如学生姓名、学号、选课代号、考试成绩。系统的输出就是希望得到什么结果，如在屏幕上显示出某班选修某门课程的所有学生的姓名和考试成绩等。系统所做的处理就是希望计算机对输入信息做什么加工，如对某班选修某门课程的所有学生的成绩进行排序，并统计成绩为优秀的学生人数等。

课程管理系统是一个相对比较简单的系统，当需要解决的问题比较复杂时，对问题的定义也就会变得非常复杂。这时需要借助于一些原则、方法和工具。有兴趣的读者可参阅一些有关软件过程学方面的书籍。

1.4.2 模块设计

需求分析阶段解决了要“做什么”的问题，设计阶段则是要解决“怎么做”的问题。设计目标就是说明系统是如何被实现的。

对复杂的问题（大型系统），理解起来总是比较困难的，这时需要将大问题分解成若干个小问题，以方便理解和解决，这就是系统的模块化。在分析阶段，已经建立了整个系统的模型，并对功能模块进行了大致的划分（层次划分），确定了系统的总体输入、处理和输出。但没有确定该怎么做。所以在设计阶段，就需要解决以下几个问题：

- 利用某种设计方法，将复杂问题划分为具体的子问题，即设计出应该包含哪些具体的功能模块。
- 确定每个模块的功能。
- 确定模块之间的关系、相互间应传递的信息。
- 确定模块之间的联系方式（接口）。
- 确定每个模块功能的实现方法和步骤（即算法）。

对于大多数软件系统，设计阶段还包括对数据结构和数据库的设计。当然，最后还需要编写设计文档。

读到这里，我们可以离开书本想一下，计算机能做什么？你可能会想计算机能做很多很多事，但事实上，计算机只能做一件事：就是执行程序。它所能完成的每项工作都是通过执行程序来实现的，只是，不同的工作需要的不同的实现方法，从而有不同的程序。所以，程序是建立在“方法”的基础上的。这里的方法，就称为“算法”。算法是实现某个具体功能的方法和步骤，是对问题处理过程的进一步细化。它不是计算机可以直接执行的，只是编写程序代码前对处理思想的一种描述，它可以是自然语言，也可以是其他的方式。

有关算法分析的详细介绍将在第 7 章呈现给读者。下面仅通过一个实例来说明算法的描述。

例 1-2 给出以下问题的算法描述：统计某个班学生的英语和数学这两门课的成绩，找出合计成绩最高并且没有不及格的学生。

算法描述如下：

步骤 1：输入全部学生姓名、学号、英语成绩、数学成绩；

步骤 2：对各个学生成绩求合计；

步骤 3：按合计对学生进行排序；

步骤 4：从排序列表中取第一位学生的成绩；

步骤 5：该学生有不及格吗？没有则打印姓名并结束；有不及格，则取下一位学生的成绩并重复步骤 5。

1.4.3 程序编码与调试

1．编码

编码就是按照需求分析和模块设计所规划好的蓝本，用真正的计算机语言去实现所规划的功能。主要涉及编码的组织及程序语言的选择。

1）自顶向下，逐步求精的设计方法

对于很小的简单的程序（如十几行的程序），程序怎么组织显得并不重要，但对于一个复杂的、成千上万行代码的大程序就不一样了（设想一下，连续 100 万行代码，如果连成一片该怎么管理），需要将其从顶向下、一步步地划分为若干个小程序块，每一个小程序块（子程序，Subprogram）完成相对单一的功能（这就像一个复杂的大问题需要逐级划分为若干个子问题一样），最终形成一个如图 1-20 所示的程序的树状结构。

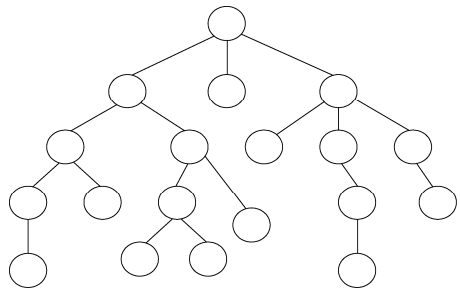


图 1-20 程序的树状结构

这种模块化的设计，可使程序的结构清晰，分工合作容易，编写和修改都比较方便。

2）程序设计语言的选择

目前，可用的计算机语言有数百种之多，每种语言的功能和性能也在不断地改进。不同的语言有不同的特点和表现形式，同样的问题用不同的语言编写出的程序也会有所不同，有时甚至会有较大的差别。有的程序语言适宜数据库的开发，有的适宜科学计算，有的针对性强，有的功能全面。针对某一具体问题，在选择程序语言时，需要考虑不同语言的适用程度及现实的可行性，如是否简单易学、语句是否容易有二义性等、是否能满足对问题求解的需要、编译程序的效率等。例如，FORTRAN、C++等语言对数值计算有更多的优势；C 语言更广泛用于操作系统和编译器的开发；在实时控制领域，汇编语言依然在广泛应用；而一些有特殊用途的语言，如 PHP，则专门用于网页设计等。

总体上，在进行程序设计时，通常从以下三个方面考虑程序语言的选择：

- **人的因素**。编程小组的人精通这门语言吗？如果不精通，需要多长时间来学习？
- **语言的能力因素**。这门语言支持所需要的一些功能吗？它能跨平台吗？它有数据库的接口功能吗？它能直接控制声卡采集声音吗？

- **其他的因素**。这门语言开发这类任务通常的开发周期是多长？这门语言是否被经常使用？

有时，可能没有多少选择，如要通过串行口控制一个外部设备，C 加上汇编语言是最明智的选择；而另外一些时候，选择会比较多。了解一些流行的语言，哪怕并不精通，对于做出合理的选择会有较大的帮助。

另外必须认识到一点的是，程序设计语言自身也是在不断发展中，如 2001 年微软公司推出的 C#语言，就既拥有 C 和 C++语言的强大功能，又具有了 Visual Bisic 易使用的特点。

2．调试

程序编制可以在计算机上进行，也可以在纸张上进行，但最终要让计算机来运行则必须输入到计算机，并经过调试，以便找出语法错误和逻辑错误，然后才能正确地运行。因此，程序调试的目的就是诊断和改正程序中可能存在的错误。

不同的语言的运行环境可能差距较大，但调试纠错这一步都是必须经过的。图 1-21 所示为一段 Visual Basic 程序在 Visual Studio2010 环境下编译器报告的语法错误。编译器在下方显示出了程序存在的错误及其位置。

一般说来，语言的检查功能只能查出语法错误，即程序是否按规定的格式书写，而逻辑错误的排查，则需要程序员自身的能力。例如，编写程序找两个数中的较大者，一不小心写成了如下的形式：

```
If a>b Then
   Tmp=a
  Else
    Tmp=a
End If
```

以上两行语句的意思：如果 a＞b，将 a 赋值给 tmp，否则，a 赋值给 tmp。这就是有逻辑上的错误，因为总是找到第一个数，使在有些情况下得不到正确结果。不幸的是，目前为止，编译程序还是检查不出此类错误。编译环境对程序的出错报告如图 1-21 所示。

图 1-21　编译环境对程序的出错报告

1.4.4 系统测试

测试是为了发现错误而执行程序的过程。它根据整个问题求解过程中各个阶段的文档（该阶段的设计要求和目标），验证系统设计的正确性。包括所有需求的功能是否被正确的实现、可靠性是否满足等。测试活动主要有以下四类：

- **单元测试**。对一个模块或几个模块组成的小功能单元做测试。
- **集成测试**。最终将本项目所有模块集成，交出完整程序产品。
- **确认测试**。验证是否与需求规格说明的描述相符。包括所有功能需求均满足；所有性能需求均达到；所有文档均已改正；其他需求已满足等。
- **系统测试**。对包括硬件、软件及其他相关设备集成在一起的测试。主要测试可恢复性、安全性、抗无意或恶意攻击的强度等。

计算机软件的整个测试过程如图 1-22 所示。

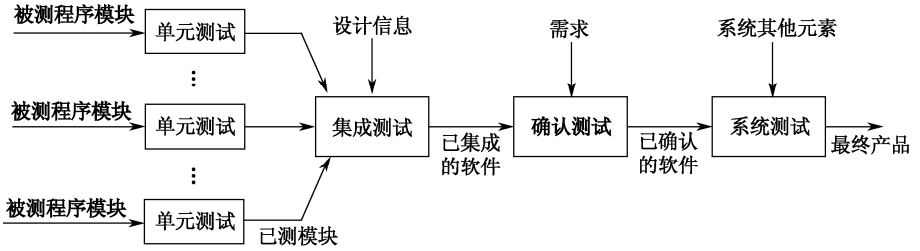


图 1-22 软件测试过程

软件测试的方法有两种：一种称为白盒测试；另一种称为黑盒测试。白盒测试的对象是程序源码，主要用于单元测试。黑盒测试作为一种方法，除单元测试外，也可用于集成测试或确认测试。

黑盒测试是对功能的测试，这种方法是将软件模块看作一个黑盒子，不关心其内部的逻辑结构，而只检查在一定的输入下，其输出是否符合功能要求。

白盒测试是测试模块内部操作是否正确，即需要测试程序的内部逻辑结构。

无论是黑盒测试还是白盒测试，都不可能将所有的输入数据拿来做测试，那样所需要的时间可能是个天文数字。例如，对图 1-23 所示的程序模块 P，使其在 32bit 字长的计算机上运行，输入 X、Y 为整数。若将所有的 X、Y 值用作黑盒测试的输入，其最大的测试数量为 $2^{32}\times2^{32}=2^{64}$。如果程序 P 测试一组 X、Y 数据需要 1ms，一天工作 24h，一年工作 365 天，则完成 2^{64} 组数据的测试需要 5 亿多年的时间。这是一个不可能完成的任务。因此，真正的测试会采用一些特定的方法，如对可能的输入数据进行分类，每一类选择几个代表作为测试数据。这样就可大大减少测试的工作量和时间。

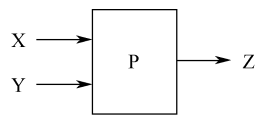


图 1-23 黑盒测试

对白盒测试类似，设计好的测试用例，将既可以达到有效测试的目的，又可提高测试的效率。有关软件测试的详细描述，可参阅其他相关书籍。

在上述整个问题求解的过程中，确定需求或说明问题的定义，是最困难的工作。对初学者来讲，由于缺乏大型系统的开发经验，甚至可能从来没有做过任何一个软件产品的设计，所以理解起来比较困难。系统测试部分也存在类似的困惑。本书将这些初学者难以理

解的内容编写出来的目的，是希望读者能够对整个问题求解的过程有基本的概念，从而对下述内容能够更好的体会。

*1.5 计算机科学研究的前沿技术

作为现代科学体系的基石之一，计算机科学已逐渐成为其他学科研究的基础，并通过与其他学科的交叉研究，演变为一种横向型科学技术，从而也使计算机科学的研究领域越来越广泛。限于篇幅，本节仅简要介绍部分计算机科学研究的新技术。

1.5.1 高性能计算

计算机在诞生的主要诱因就是数值计算。科学计算是计算机最初的主要甚至唯一的功能。第一台电子计算机完成一次加法运算的时间约需 0.2ms，这在当时已是很了不起的高速度了。但随着信息社会的发展，人类对信息处理能力的要求越来越高，不但科学研究、石油勘探、气象预报、金融保险、航空航天等需要高速的数据处理能力，甚至网络游戏都对高性能计算有了需求。现在，作为计算机科学的一个研究分支，高性能计算及高性能计算机已成为信息领域前沿技术，在保障国家安全、推动国防科技进步、促进尖端武器发展方面具有直接推动作用，是衡量一个国家综合实力的重要标志之一。

1．什么是高性能计算

高性能计算（High Performance Computing）主要研究并行[1]算法、并行软件技术及系统体系结构，并致力于研制高性能计算机（High Performance Computer）。

从字面意思看，“高性能计算机”就是性能指标高的计算机。计算机的哪些指标要达到什么程度才能称为“高”呢？如同计算机技术在不断发展一样，“高性能计算机”的衡量标准也在不断变化中，目前，主要以计算速度（尤其是浮点运算速度）作为标准。

高性能计算的发展始于 20 世纪 80 年代，以数值模拟为主体。整个 80 年代，在美国国家科学基金会、美国航空航天局及美国政府的支持下，高性能计算机的研制得到快速发展，促进了峰值速度可达每秒 10 亿次的 Cray 超级矢量计算机和多 CPU 并行计算机等高性能计算机的问世。从 90 年代起，美国先后推出了“高性能计算和通信”计划和“先进模拟和计算”计划。重点是高性能计算机系统、并行算法与先进软件技术、基础研究和教育网络，以及以科学计算为基础的核武器库存可靠性和有效性研究等。

图 1-24 天河一号 A 超级计算机

我国高性能计算机的研制历经几十年努力也得到很大发展。目前，我国已成为继美国和日本之后，第三个具备高性能计算机系统能力的国家，如 2010 年由国防科技大学研制的“天河一号 A”的实测运算能力每秒达 2507 万亿次，成为当年 5 月份世界最快的超级计算机，如图 1-24 所示。如今，高性能计算在我国的核武器模拟、气候变化分析和模拟、量子模拟等重要科技领域已得到有效应用，并取得了十分重要的成果。

[1] 并行（Parallelism）是指多个任务在同一时刻同时运行。

2. 高性能计算机的关键技术

高性能计算机的研究涉及软硬件/技术、通信技术、纳米技术等多个学科，近年主要集中于研究大规模并行处理体系结构、高性能算法、可重构计算和功耗等方面。

大规模集成电路技术对计算机性能的提高无疑有很大的影响，而体系结构的优劣同样是影响计算性能的重要因素。先看一些生活中随处可见的案例：

> ① 节日的高速公路上，因某辆车出现故障而造成的数公里车辆拥堵。
>
> ② 干旱时排队接水的人形成长龙，水却流得很慢。
>
> ③ 虽然人很多，但工作任务却只分给了少数几个人。结果忙的忙“死”，闲的闲“死”。
>
> ……

生活中的这些问题反映到计算机系统中，就是体系结构问题。由于存储器的存取速度和处理器的计算能力之间存在较大的不平衡，使处理器可以进行高速计算，但计算需要的数据存储器却来不及提供（如上例中的①和②），即数据访问与计算能力存在较大差距[1]。

实现高性能计算的手段之一是并行处理多个任务。这里的并行是指将计算任务分配到系统中的各个计算资源上，使之同时工作，以使整体计算能力得到充分发挥。能否保证每个计算部件都既无空闲或等待，也不存在过负载（如上例中的③），这是高性能计算中又一个很重要的研究课题——负载均衡。负载不均衡将会导致计算效率的大幅下降。

每秒千万亿次的计算系统需要并行度和并行效率更高的算法。随着所求解问题规模的增大，需要新的物理模型，反过来又需要不同的求解方法。与软/硬件性能提高相比，算法改进对应用问题求解的性能影响会更加显著，因此，面向应用问题进行的高性能算法设计也是超级计算机应用的重要基础。除高效算法外，高性能计算机在软件技术的研究方面还包括操作系统、面向新一代体系结构的编译优化和实现技术等。

目前的高性能计算机的主流体系结构都是基于冯·诺依曼的计算机理论基础。传统计算机体系结构存在如存储器墙、I/O 通信、功耗、程序复杂性等问题。随着技术的发展，现代计算机系统规模不断增大，系统结构越来越复杂，创新高性能计算体系结构已是大势所趋。

1.5.2 普适计算

只有当机器进入人们生活环境而不是强迫人们进入机器世界时，机器的使用才能像林中漫步一样新鲜有趣。

案例 1：在一个智能教室环境下，如果投影设备的显示效果不是很理想，教师可以通过自己的掌上电脑向学生的掌上电脑发送电子课件。当教师走近学生讨论组时，其掌上电脑会动态加入该组，下载该组正在讨论的材料。

这就是一个普适环境，它由投影机、教师掌上电脑、学生掌上电脑组成，该系统通过可重新配置的上、下文敏感中间件，突出对环境的感知和动态自组网络通信的支持。

案例 2：一个普适医疗服务系统，可以提供任何时间、任何地点的医疗服务访问。在一辆急救车上配备无线定位系统，就可准确定位突发事故现场，同时利用无线网络获取实

[1] 这一问题称为存储器墙。

时的交通信息。另外，在事故现场，通过便携式或移动式设备监测病人的脉搏、血压、呼吸等数据，通过无线网络访问分布式的医疗服务系统，下载有关病历数据等必要信息。

除了基于定位系统的应急响应机制，普适医疗服务系统的功能还包括基于移动设备和无线网络的远程医疗诊断、远程病人监护，以及远程访问具有患者病历信息的医疗数据库。

施乐公司 Palo Alto 研究中心的首席技术官 Mark Weiser 曾经说过，最深刻和强大的技术应该是“看不见”的技术，是那些融入日常生活并消失在日常生活中的技术。这个被称为“普适计算之父”的人在 20 世纪 90 年代初就声称：受社会学家、哲学家和人类学家的影响，他重新审视了网络计算模式。他指出，21 世纪的计算将是一种无所不在的计算（Ubiquitous Computing）。

因此，普适计算又称为普存计算、普及计算（Pervasive Computing 或者 Ubiquitous Computing），强调将计算和环境融为一体，而让计算本身从人们的视线中消失，使人的注意力回归到要完成的任务本身。它的含义：在普适计算的模式下，人们能够在任何时间、任何地点以任何方式进行信息的获取与处理。或简单地说，是一种无处不在的计算模式。

互联网应用的兴起使计算模式继主机计算和桌面计算之后进入一种全新的模式，也就是普适计算模式。这种新的计算模式强调把计算机嵌入到人们日常生活和工作环境中，使用户能方便地访问信息和得到计算的服务。

普适计算当然包括移动计算，但普适计算更强调环境驱动性。这要求普适计算对环境信息具有高度的可感知性，人—机交互更自然化，设备和网络的自动配置和自适应能力更强，所以，普适计算的研究涵盖传感器、人—机交互、中间件、移动计算、嵌入式技术和网络技术等领域。

1.5.3 云计算

云计算（Cloud Computing）概念是由 Google 提出的，是分布式计算、并行计算和网格计算的发展，或者说是这些科学概念的商业实现，指通过网络以按需、易扩展的方式获得所需的服务。

云计算的核心思想，是将大量用网络连接的计算资源统一管理和调度，构成一个计算资源池向用户按需服务。提供资源的网络被称为“云”。“云”中的资源在使用者看来是可以无限扩展的，并且可以随时获取，按需使用，随时扩展，按使用付费。这就像用水电一样付费使用。

云计算的基本原理是通过使计算分布到大量的分布式计算机上（而非本地计算机或远程服务器中），使得企业能够将资源切换到需要的应用上，根据需求访问计算机和存储系统。

在云计算模式下，用户不再需要购买复杂的硬件和软件，而只需要支付相应的费用给“云计算”服务提供商，通过网络就可以方便地获取所需要的计算、存储等资源。从服务的角度来看，云计算是一种全新的网络服务模式，将传统的以桌面为核心的任务处理转变为以网络为核心的任务处理，利用互联网实现自己想完成的一切处理任务，使网络成为传递服务、计算力和信息的综合媒介，真正实现按需计算、网络协作。

1.5.4 人工智能

人工智能（Artificial Intelligence）是研究、开发用于模拟、延伸和扩展人的智能的理论、

方法、技术及应用系统的一门新的技术科学，是计算机科学的一个分支，它企图了解智能的实质，并生产出一种新的能以人类智能相似的方式做出反应的智能机器。

人工智能的基本研究内容主要包括以下几个方面。

（1）机器感知。主要包括计算机视觉和计算机听觉，研究用计算机来模拟人和生物的感官系统功能，使计算机具有“感知”周围世界的能力；具体来说，就是让计算机具有对周围世界的空间物体进行传感、抽象、判断的能力，从而达到识别、理解的目的。根据其处理过程的先后及复杂程度，计算机视觉的任务可以分成下列几个方面：图像的获取、特征抽取、识别与分类、三维信息理解、景物描述和图像解释。计算机听觉建立在机器识别语言、声响和自然语言理解的基础上。语言理解包括语音分析、词法、句法和语义分析。机器感知是计算机获取外部信息的基本途径，是使机器具有智能不可缺少的组成部分，对此人工智能中已经形成两个专门的研究领域：模式识别和自然语言理解。

（2）机器思维。指计算机对通过感知得来的外部信息及其内部的各种工作信息进行有目的的处理。正像人的智能来源于大脑的思维活动一样，机器智能也是通过机器思维实现的，因此，机器思维是人工智能研究中最重要、最关键的部分。为了使计算机能模拟人类的思维活动，需要开展以下几个方面的研究：

- 知识的表示，特别是各种不精确、不完全、非规范知识的表示。
- 知识的组织、累积和管理技术。
- 知识的推理，特别是各种不精确推理、归纳推理、非单调推理、定性推理。
- 各种启发式搜索及控制策略。
- 神经网络、人脑的结构及其工作原理。

（3）机器学习。学习是人类具有的一种重要智能行为，人类能够获取新的知识、学习新技巧，并在实践中不断完善、改进。机器学习就是要使计算机具备这种学习能力，在不断重复的工作中对本身能力的增强或者改进，使得在下一次执行同样任务或类似任务时，会比现在做得更好或效率更高，并且能克服人类在学习中的局限性，如遗忘、效率低、注意力分散等。

（4）机器行为。与人的行为能力相对应，机器行为主要是指计算机的表达能力，如“说”、“写”、“画”等。对于智能机器人，它还应具有人的四肢功能，能走路、能操作。

（5）智能系统及智能计算机构造技术。人工智能的最终目标就是要构造智能系统及智能机器，因此，需要开展对系统分析与建模、构造技术、建造工具及语言的研究。

1950 年，计算机理论的奠基人图灵在哲学性杂志《精神》上发表了一篇题为“计算机和智能”（Computingmachiery and Intelligence）的著名文章，文章提出了一个检验计算机是否具备人类“思维”的方法，后来被称为“图灵测试”或“图灵检验”。

被测试者一个是人，另一个是声称有人类智力的机器。测试时，测试人与被测试人分开，测试人通过一些装置（如键盘）向被测试者提出问题，这些问题可以是任何问题。提问后，如果测试人能够正确地分出谁是人谁是机器，那机器就没有通过图灵测试，如果测试人没有分出，则这个机器就是有人类智能的。

当然，目前还没有一台机器能够通过图灵测试，也就是说，计算机的智力与人类还相差很远。但图灵指出：“如果机器在某些现实的条件下，能够非常好地模仿人回答问题，以至于提问者在相当长时间里误认为它不是机器，那么机器就可以被认为是能够思维的”。

虽然还没有成功通过图灵测试的计算机，但已有计算机在测试中“骗”过了测试者。著名的“深蓝（DeepBlue）”机器人就是一个很好的例证。1997 年 5 月 11 日，由 IBM 公司研制的、起名为“深蓝（DeepBlue）”的超级计算机 AS/6000 SP，与“人类最伟大的棋手”前苏联国际象棋世界冠军卡斯帕洛夫进行的人—机象棋大赛，计算机以微弱优势取胜。

这个案例及众多的影视作品都不仅会让人设想：未来会出现能够骗过大多数人的计算机吗？

1.5.5 物联网

物联网（The Internet of Things），顾名思义，就是“物物相连的互联网”，是新一代信息技术的重要组成部分。它是通过射频识别（Radio Frequency Identification，RFID）、红外感应器、全球定位系统、激光扫描器等信息传感设备，按约定的协议，把任何物体与互联网相连接，进行信息交换和通信，以实现对物体的智能化识别、定位、跟踪、监控和管理的一种网络。

物联网的核心和基础仍然是互联网，是在互联网基础上延伸和扩展的网络。其用户端可延伸和扩展到任何物体与物体之间，实现物体与物体之间的信息交换和通信。

物联网可分为三层：感知层、网络层和应用层。

感知层由各种传感器（如温度传感器、湿度传感器、摄像头、GPS 等感知终端）和传感器网关构成。其作用相当于人的眼、耳、鼻、喉和皮肤等神经末梢，它是物联网识别物体，采集信息的来源，其主要功能是识别物体，采集信息。

网络层由各种私有网络、互联网、有线和无线通信网、网络管理系统和云计算平台等组成，相当于人的神经中枢和大脑，负责传递和处理感知层获取的信息。

应用层是物联网和用户（包括人、组织和其他系统）的接口，它与行业需求结合，实现物联网的智能应用。

目前，物联网技术已在多个行业领域得到应用，如上海浦东国际机场的入侵防护系统。为了保护机场安全，铺设了 3 万多个传感结点，覆盖了地面、栅栏和低空探测，可以防止人员的翻越、偷渡、恐怖袭击等攻击性入侵。

物联网技术近年来发展迅速，已广泛应用于物流、零售、制药和安保等各个领域，在不断地改变着我们目前的生活方式。未来的物联网将会向更加智能化的方向发展。

习题

一、填空题

1．计算机科学是主要研究（　　）、（　　）和（　　）的学科。

2．在模型建立的前提下，利用计算机求解问题的核心工作就（　　）设计。

3．算法是一组规则，它的主要特性是（　　）、（　　）、（　　）、（　　）和（　　）。

4．要使一个问题能够用计算机解决，其必要条件是（　　）。

5．在计算机内，一切信息都是以（　　）形式表示的。

6．如果说图灵机 A 能够完全模拟图灵机 B，则意味着（　　）。如果 A 和 B 能够相互模拟，则表示（　　）。

7．图灵机中的纸带可以相当于计算机中的（　　）。

8．第一代计算机的主要部件是由（　　）构成的。

9．未来全新的计算机技术主要指（　　）、（　　）和（　　）。

10．未来电子计算机的发展方向是（　　）、（　　）、（　　）和（　　）。

11．目前，国际上广泛采用的西文字符编码是标准（　　），它是用（　　）位二进制码表示一个字符。

12．采用 16 位编码的一个汉字存储时要占用的字节数为（　　）。

13．位图文件的存储格式为（　　），用数码相机拍摄的照片的文件格式一般为（　　）。

14．若处理的信息包括文字、图片、声音和电影，则其信息量相对最小的是（　　）。

15．模拟信号是指（　　）都连续变化的信号。

16．计算机中对信息的组织和管理方式有两种，即（　　）和（　　）。

17．软件的测试方法包括（　　）和（　　）。

18．普适计算的主要特点是（　　）。

二、简答题

1．简述计算机采用二进制的原因。

2．图灵机模型主要由哪 4 个部分组成？

3．图灵机在形式上可以用哪 7 个元素描述？它们分别表示什么含义？

4．图灵机模型中的 4 个要素是什么？

5．简述图灵机的工作过程。

6．简述问题求解的一般过程。

7．简述基于计算机的信息处理的一般过程。

8．简述高性能计算机涉及的主要关键技术。

第 2 章　计算机基础知识

引言

计算机是 20 世纪人类最伟大的发明之一，在从诞生起至今的半个多世纪中，它由最初的“计算”工具，迅速发展成为应用于各行各业的信息处理设备，成为人类工作和生活中不可缺少的助手。现在，几乎每个年轻人都会使用计算机。但每位使用者是不是都了解计算机呢？本章将从计算机的基本组成入手，先为读者描述什么是计算机，然后介绍编写程序时需要涉及到的计算机中的常用计数制，以及计算机中二进制数的表示和运算。在第 1 章中已介绍二进制是计算机自身唯一采用的计数制。

教学目的

- 理解计算机系统的逻辑构成和物理构成。
- 理解计算机中常用计数制的表示及其相互间的转换。
- 了解二进制数的表示。
- 理解机器数的表示。
- 掌握二进制数的算术运算和逻辑运算。
- 了解基本逻辑门的表示及其真值表。

2.1　计算机系统

我们平时说的计算机，确切地说是计算机系统。因为没有人会认为不带显示器和键盘、鼠标的机器能称为“计算机”。其实，就计算机系统而言，它可以说是一个“广义”的概念。因为现代的计算机系统，巨型机和微型机还是有比较多的差别，若融入网络技术和辅助的软件技术，如并行机、阵列机、机群系统等，而第 1 章介绍到的云计算，宏观上讲也可以称为一个系统。

2.1.1　计算机系统构成

计算机系统与计算机、微处理器是从宏观到微观的三个不同的层次。计算机系统不仅包含物理上能够看得见的硬件实体，还包含运行于实体之上的、可实现各种操作功能的软件。图 2-1 所示为计算机系统的概念结构。

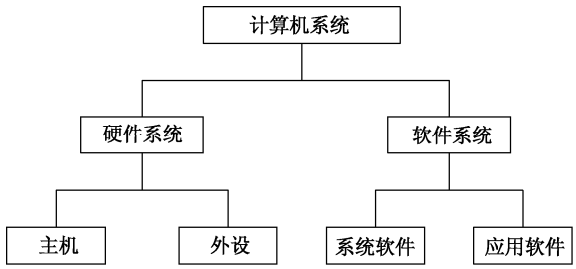


图 2-1　计算机系统的概念结构

1．硬件系统

微机硬件系统包括主机和能够与计算机进行信息交换的外部设备两部分。主机位于我们日常看到的主机箱内，主要包括微处理器（CPU）、内存储器、I/O 接口、总线和电源等。其中，微处理器是整个系统的核心。能否与处理器进行直接信息交换是能够成为主机部件的重要标志。“直接信息交换”，就是不需通过任何中间环节（用专业术语说是接口），就能够实现从处理器接收数据或向处理器发送数据。典型的如内存，与处理器间的数据传输就是直接进行的。事实上，计算机正在运行的所有程序和数据，不论其曾经存放在哪里，在运行前都必须送入内存，这样才能保证计算机工作的高速度。这一点，将在后续内容中逐步介绍。

今天，如果有人告诉你他买了一台计算机，你一定会清楚他不是只抱了一台主机箱回来，而至少还包括显示器、键盘和鼠标。这些称为计算机的基本外部设备。

外部设备，是指所有能够与计算机进行信息交换的设备（当然，这种信息交换需要通过接口）。既包括上述操作计算机所必须的基本外部设备，也包括其他各种能够连接到计算机的仪器。我们将用于向计算机输入信息的设备称为输入设备，如键盘、鼠标器、扫描仪等；接收计算机输出信息的设备则称为输出设备，如显示器、打印机、绘图仪等。当然，也有些设备既能接收计算机输出的信息，也能向系统输入信息，如数码摄像机、硬磁盘等。亦即它们兼具了输入设备和输出设备的功能，具体担当何种角色，则视其在某个时刻传送数据的方向。

相对于主机，外部设备的主要特点就是不能与处理器直接进行数据输入和输出，数据的传输必须通过接口，如硬磁盘，虽然安装在主机箱内，但不属于主机系统，因为它与处理器的通信需要通过专用接口进行。

有关计算机常用外设的基本工作原理见附录 A。

2．软件系统

硬件系统是计算机工作的物理基础，但要使其正常工作并完成各种任务，还必须要有相应的软件支撑。软件，不仅是一般概念中的程序，而是程序、数据及相关文档的总称。这里，数据是程序处理的对象，文档是指与程序开发、维护和使用有关的各种图文资料。软件可以分为两大类：系统软件和应用软件。

系统软件是管理、监控和维护计算机软硬件资源的软件，由计算机设计者提供，包括操作系统和各种系统应用程序。操作系统（Operating System，OS）是配置在计算机硬件上的第一层软件，是其他软件运行的基础。其主要功能是管理计算机系统中的各种硬件和软

件资源（如存储器管理、文件管理、进程管理、设备管理等），并为用户提供与计算机硬件系统之间的接口（如通过键盘发出命令控制作业运行等）。在计算机上运行的其他所有的系统软件（如编译程序、数据库管理系统、网络管理系统等）及各种应用程序，都要依赖于操作系统的支持。因此，操作系统是计算机中必须配置的软件，在计算机系统中占据着极其重要的位置。目前，较为流行的操作系统有 Windows 系列、UNIX、Linux 等。

系统应用程序运行于操作系统之上，是为应用程序的开发和运行提供支持的软件平台，主要包括以下方面：

- 各种语言及其汇编或解释、编译程序。用于将汇编语言或各种高级语言编写的程序翻译成计算机硬件能够直接识别的用二进制码表示的机器语言。计算机硬件是由各种逻辑器件构成的，只能识别电脉冲信号，也就是“0”和“1”组成的二进制码，这种由二进制码组成的计算机语言称为机器语言，人类很难理解和记忆。目前，广泛使用的计算机程序设计语言都是接近人类自然语言的高级语言，为了使计算机能够理解，必须要经过一个翻译的过程，而这类程序的功能就是实现这样的翻译。
- 计算机的监控管理程序（Monitor）、故障检测和诊断程序及调试程序（Debug）。它们负责监控和管理计算机资源，并为应用程序提供必要的调试环境。
- 各类支撑软件，如数据库管理系统及各种工具软件等。

应用软件是应用程序员利用各种程序设计语言编写出的、面向各行各业实现不同功能的应用软件，如工程设计程序、数据处理程序、自动控制程序、企业管理程序等。目前，软件的设计还没有摆脱手工操作的模式，但随着软件技术的进步，应用软件也在逐渐地向标准化、模块化方向发展，目前，已形成了部分用于解决某些典型问题的应用程序组合，称为软件包（Package）。

软件系统的核心是系统软件，而系统软件的核心则是操作系统。

计算机系统是硬件和软件的结合体，硬件和软件相辅相成，缺一不可。硬件是计算机工作的物质基础，而软件是计算机的灵魂。没有硬件，软件就失去了运行的基础和指挥对象；而没有软件，计算机就不能工作，其效能就不能充分发挥出来。

对某项具体任务，通常既可以用硬件完成，也可以通过软件完成。从理论上讲，任何软件算法都能用硬件实现，反之亦然，这就是软件与硬件的逻辑等价性。设计计算机系统或是在现有的计算机系统上增加功能时，具体采用硬件还是软件实现，取决于价格、速度、可靠性等因素。早期的计算机受技术和成本的限制，硬件都相对简单。如今，随着超大规模集成电路技术的发展，以前由软件实现的功能现在更多地直接用硬件实现，为的是提高系统的运行速度和效率。另外，在软件和硬件之间还出现了固件（Firmware），它们在形式上类似于硬件，但从功能上又像软件，可以编程和修改，这种趋势称为软件的硬化和固化。

2.1.2 微型计算机主机板

我们日常接触最多或者说是唯一接触的计算机是微型计算机（Microcomputer），又称为个人计算机（Personal Computer，PC）。因此，这里就以微型机为例，介绍微机的物理构成——系统板（Systemboard），又称为主机板（Mainboard）。

主机板是微机最基本的也是最重要的部件之一，在整个微机系统中扮演着举足轻重的角色。可以说，主机板的类型和档次决定着整个微机系统的类型和档次，主机板的性能影

响着整个微机系统的性能。

主机板在结构上主要有 AT 主板、ATX 主板、NLX 主板和 BTX 主板等类型。它们之间的区别主要在于各部件在主板上的位置排列、电源的接口外形、控制方式及尺寸等。不论哪种结构，均采用开放式结构。可以通过更换安装在扩展插槽上的外围设备控制卡（适配器），实现对微机相应子系统的局部升级。图 2-2 为一个实际的 ATX 主板的布局结构及外形图。

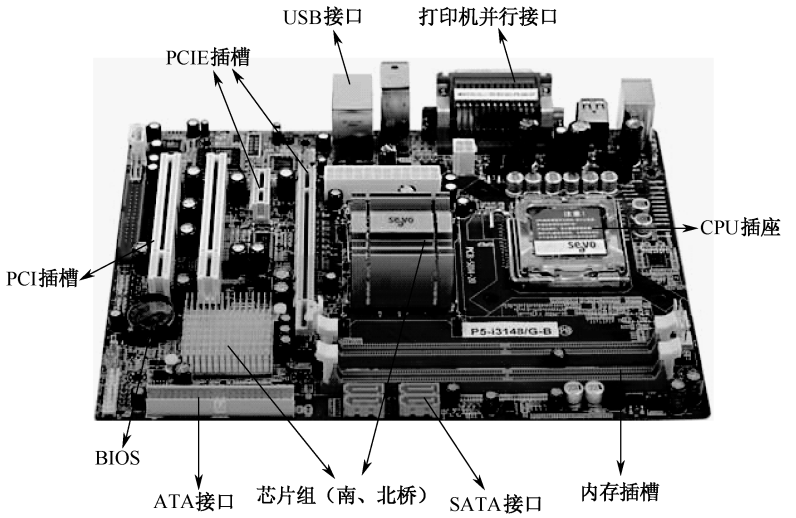


图 2-2　ATX 主板的布局结构及外形图

主板位于主机箱内，上面安装了组成计算机的主要电路系统，主要包括芯片部分、扩展槽和对外接口三种类型的部件。

1．芯片部分

这部分除微处理器（CPU）外，主要有控制芯片组和 BIOS。

芯片组是主板上一组超大规模集成电路芯片的总称，是主板的关键部件，用于控制和协调计算机系统各部件的运行，它在很大程度上决定了主板的功能和性能。可以说，系统的芯片组一旦确定，整个系统的定型和选件变化范围也就随之确定。

典型的芯片组由北桥芯片和南桥芯片两部分（两片芯片）组成，又称为南、北桥芯片。图 2-2 中 CPU 插槽旁边被散热片盖住的就是北桥芯片。北桥芯片是芯片组的核心，主要负责处理 CPU、内存、显卡三者间“交通”，由于发热量较大，故需加装散热片散热。南桥芯片主要负责硬盘等存储设备和 PCI 之间的数据流通。

需要说明的是，现在一些高端主板上已将南、北桥芯片封装到一起，使“芯片组”在形式上只有一个芯片，提高了芯片组的功能。

BIOS 又称为系统 BIOS，是一块方块状的存储器芯片，里面存有与该主板搭配的基本输入/输出系统程序。能够让主板识别各种硬件，还可以设置引导系统的设备，调整 CPU 外频等。BIOS 芯片是可读/写的只读存储器（EPROM 或 E^2PROM）。机器关机后，其上存储的信息不会丢失。在需要更新 BIOS 版本时，还可方便地写入。当然，不利的一面便是会让主板遭受病毒的袭击。

系统 BIOS 程序主要包含以下几个模块：

- 上电自检（Power-On Self Test，POST）。在微机加电后，CPU 从地址为 0xFFFFFF0H 处读取和执行指令，进入加电自检程序，测试整个微机系统是否工作正常。
- 初始化。包括可编程接口芯片的初始化；设置中断矢量表（一个专门用于存放中断程序入口地址的内存区域）；设置 BIOS 中包含的中断服务程序的中断矢量（即将这些中断程序入口地址放入中断矢量表中）；通过 BIOS 中的自举程序将操作系统中的初始引导程序装入内存，从而启动操作系统。
- 系统设置（Setup）。装入或更新 CMOS RAM 保存的信息。在系统加电后尚未进入操作系统时，按 Del 键（或其他热键）可进入 Setup 程序，修改各种配置参数或选择默认参数。

2．扩展槽

安装在扩展槽上的部件属于“可插拔”部件。“可插拔”是指这类部件可以用“插”来安装，用“拔”来反安装。主板上的扩展槽包括内存插槽和总线接口插槽两大类。内存插槽一般位于 CPU 插座下方，用于安装内存储器（又称为内存条，如图 2-3 所示）。通过在内存插槽上插入不同的内存条，就可方便地构成所需容量的内存储器。主板上内存插槽的数量和类型对系统主存的扩展能力及扩展方式有一定影响。现在主板上大多采用 184 线的内存插槽，配置的内存条也必须是 184 个引脚。

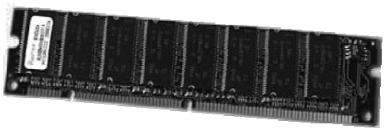


图 2-3　内存条

总线接口插槽是 CPU 通过系统总线与外部设备联系的通道，系统的各种扩展接口卡都插在总线接口插槽上。总线接口插槽主要有 PCI 插槽、AGP 插槽或 PCI Express（PCIE）插槽。PCI 插槽多为乳白色，是主板的必备插槽，可以插入声卡、股票接受卡、网卡、多功能卡等设备。AGP 插槽的颜色多为深棕色，位于北桥芯片和 PCI 插槽之间，用于插入 AGP 显卡，有 1×、2×、4×和 8×[1]之分。在 PCI Express 出现之前，AGP 显卡是主流显卡，其传输速度最高可达到 2133MB/s(AGP8×)。

随着三维性能要求的不断提高，AGP 总线的传输速度已越来越不能满足视频数据处理的要求。在目前的主流主板上，显卡接口多转向 PCI Exprss。PCI Exprss 插槽有 1×、2×、4×、8×和 16×之分。

3．对外接口

在微机主机板上配置有各类接口，用于理解包括硬磁盘在内的各种外部设备。硬盘接口用于理解硬磁盘，类型有 IDE（Integrated Drive Electronics，电子集成驱动器）接口、SATA（Serial Advanced Technology Attachment，串行高级技术附件）接口等。在型号老些的主板上，多集成两个 IDE 接口，通常 IDE 接口都位于 PCI 插槽下方。在现代新型主板上，IDE 接口大多缩减，甚至没有，代之以 SATA 接口。

SATA 是由 Intel、IBM、Dell、APT、Maxtor 和 Seagate 公司共同提出的硬盘接口规范，它首次将硬盘的外部传输速率理论值提高到 150MB/s，比之前的并行传输的 ATA/133 高出

[1]　n×表示 n 倍速，即对原来的时钟脉冲进行技术处理后，使时钟频率变成 n 倍频。

约 13%，而且还将进一步扩展到 2×和 4×（300MB/s 和 600MB/s）。SATA 通过提升时钟频率来提高接口传输速率，使串行接口硬盘的传输速度大大超过并行。

除硬盘接口外，还有用于连接各种外部设备的串行和并行接口插座，包括 RS-232 串行口插座、USB（Universal Serial Bus，通用串行总线）插座及标准并行口插座（EPP 或 ECP 规范）。

COM（Component Object Model）接口是串行接口，目前，大多数主板都提供 COM1 和 COM2 的两个 COM 接口，作用是连接串行鼠标和外置 Modem 等设备。COM2 接口比 COM1 接口具有优先响应权。

PS/2 接口是专用于连接键盘和鼠标的串行接口，比 COM 接口的传输速率稍快。一般情况下，鼠标的接口为绿色、键盘的接口为紫色。虽然现在绝大多数主板依然配备该接口，但支持该接口的鼠标和键盘越来越少，而逐渐被 USB 接口取代。

USB 接口是目前最为流行的外设接口，最大可以支持 127 个外设，并且可以独立供电，应用非常广泛。USB 接口可以从主板上获得 500mA 的电流，支持热拔插，真正做到了即插即用。目前的 USB2.0 标准最高传输速率可达 480Mb/s。USB3.0 已经开始出现在最新主板中，不久将会被推广。

老式主板上还有用于连接打印机或扫描仪的并行接口 LPT。但随着 USB 技术的发展，现在使用 LPT 接口的打印机和扫描仪已经很少，基本都被 USB 接口所取代。

除上述这些主要部件外，主板上还有用于连接硬盘、光驱等的电缆插座、键盘/鼠标接口，以及许多不可缺少的逻辑部件和跳线开关等。所有这些部件的密切联系、相互沟通，实现了整个微型机中各部件间的数据交流。

2.1.3　计算机的主要性能指标

表征微机系统性能的指标较多，这里简要介绍其中的几项。

1．主频

主频是主时钟频率的简称，单位为兆赫兹（MHz）。指在一秒种内发生的同步脉冲数。主频很大程度上决定了计算机的运行速度，主频越高意味着计算机的速度也越快。

2．运算速度

程序由一条条指令组成（见 3.2 节），执行一条指令所花费的时间越少，计算机的工作速度就越快。衡量计算机针对整数的运算速度用 MPS（Million Instructions Per Second，每秒百万条指令）表示，对于浮点运算，一般使用 MFLOPS（Million Floating Point Operations Per Second）表示，即每秒百万次浮点运算。

3．内存容量

内存容量指内存存储数据的能力。存储容量越大，CPU 能直接访问到的数据就越多。存储器最基本的计量单位是字节（Byte），一个字节由一个 8 位（bit）二进制数组成，简称 1B（Byte），此外还有 KB、MB、GB 和 TB 等表示方式。

4．字长

字长指 CPU 能够同时处理的二进制位数。字节越长，运算精度越高，数据处理速度越快。

5．外部设备的配置及扩展能力

外部设备的配置及扩展能力主要指计算机系统连接各种外部设备的可能性、灵活性和适应性。常见配置有 C 盘驱动器的配置、硬盘接口类型与容量、显示器的分辨率等。

2.2 计算机中的数制

由第 1 章的描述已知，计算机硬件能够直接识别只有由“0”和“1”构成的二进制码，也就是说，计算机中的数是用二进制表示的。

但用二进制数表示一个较大的数时，既冗长又难以记忆，人们在日常生活中仍习惯于使用十进制数。借助于软件的辅助，现代计算机也能够间接理解十进制等一些其他的进制数。所以，在学习计算机的基本原理和应用之前，需要首先了解和掌握计算机中数的表示、常用的计数制，以及如何实现它们相互间的转换。

2.2.1 数的编码单位

现代计算机中全部采用 0 和 1 来表示各种信息，不同的信息具有不同的长度。在学习计算机中的常用计数制之前，我们先要了解计算机中信息的编码长度。

1．数位

计算机中的一个 0 或 1 是信息的最小单位，称为位（Binary Digit，bit），或简写为 b。它表示逻辑器件的一种状态：“断开”或“闭合”。在内存储器中，它可以是用于存放信息的晶体管的“开”或“关”，也可以是某个电容的充电或放电；在硬磁盘中，bit 通过磁盘盘片表面的磁场方向表示（“南—北”或“东—西”）；在常用的 CD-ROM 光盘上，它是光的反射与否；而在计算机所处理和存储的数字音频信号中，可以用“1”表示高音，用“0”表示低音。

一个十进制数可以由多个数位构成，如 128，就是由 3 个数位构成的，最右边是个位，也是这个十进制整数的最低位，其权值是 10^0。之后，从右向左，依次是十位（次低位）和百位（最高位），相应的权值分别是 10^1 和 10^2。

同样，二进制数的位因其处于数的不同位置也具有不同的权值，其权值的大小也是从右向左依次增加的。如二进制数 1011，同样最右边是最低位，权值为 2^0；之后从右向左，其权值分别为 2^1、2^2，最高位的权值为 2^3。由于最低位的权值是 2^0，因此，在计算机中，常用 bit0 来表示一个二进制数的最低位，高位则依次为 bit1，bit2，……。

对一个二进制数，哪一位是最低位，哪一位是最高位，且最低位称为“第 0 位”（不是“第 1 位”），这些是初学者必须要清楚的问题。

2．字节

一位 0 或 1 无法表示太多数据，需要将多位组合起来。由于计算机对数据的处理多以 8 位二进制码（或 8 位的整数倍）为单位，所以，常将 8 位二进制码作为一个整体，称为 1B（Byte），简写为 B。1B 是 8 位二进制码，能够表示的最大数是 $2^8-1=255$。

字节是计算机中表示存储空间大小的基本容量单位。例如，计算机内存的存储容量、

磁盘的存储容量等都是以字节为单位表示的。此外，为表示更大的数字，我们将更多字节结合起来，如 2 字节是 16 位，能够表示的最大数就是 $2^{16}-1=65535$。以此类推，就有了以下这些表示大数据的单位：千字节（Kilobyte，KB），兆字节（MB，Megabyte），十亿字节（Gigabyte Byte，GB），万亿字节（Terabyte，TB）等。它们之间的换算关系为

1B=8bit

$1KB=2^{10}$ B=1024B

$1MB=2^{10}KB=2^{20}$ B=1024KB

$1GB=2^{10}MB=2^{20}$ KB=2^{30} B=1024MB

$1TB=2^{10}GB=2^{20}MB=2^{30}$ KB=2^{40} B=1024GB

3．字长

在计算机诞生初期，受各种因素限制，计算机一次能够同时（并行）处理 8bit 二进制码，即一次能够进行 8 位二进制码的加减乘除等运算。随着电子技术的发展，计算机的并行能力越来越强，从 8 位、16 位、32 位，直至今天，微型机的并行处理能力一般为 64 位，大型机已达 128 位。我们将计算机一次能够并行处理的二进制位数称为该机器的字长，又称为计算机的一个“字”。因此，早期的计算机被称为 8 位机，而今天我们常用的个人计算机（Personal Computer，PC）则称为 64 位机。

字长是计算机的一个重要性能指标，直接反映了一台计算机的计算能力和精度。字长越长，计算机处理数据的速度就越快。这点可以通过一个例子来说明，如计算 5×8，我们可以立即得出答案为 40。但如果要计算 55×88，就不可能立即得到正确的答案。这是因为 55×88 的运算已超出了人脑的“字长”。为了得出结果，需要将复杂的问题（如 55×88）进行分解，如分解为(50×80)+(50×8)+(5×80)+(5×8)。这样，虽然较容易得出结果，但可以看出，需要花费比较多的时间。随着数字的增大，需要花的时间就越长。

人脑是这样，计算机同样是这样的。计算机能够一次直接处理的最大数决定于计算机的字长。如果要计算的数据超出了计算机的字长，就必须对数据进行分解。一台字长为 16 位的计算机，可以直接处理 2^{16}（65536）之内的数据，对于超过 65536 的数就必须分解之后才能处理。32 位机比 16 位机优越的原因就在于它在一次操作中能处理的数更大（2^{32}，达 40 亿）。能处理的数字越大，则操作的次数就越少，系统的效率也就越高。

2.2.2　计算机中的常用计数制

1．十进制数

十进制数有 0～9 十个数字符号，用符号 D 标识。一个任意十进制数可用权展开式表示为

$$(D)_{10}=D_{n-1}\times10_{n-1}+D_{n-2}+\cdots+D_1\times10^1+D_0\times10^0+D_{-1}\times10^{-1}+\cdots+D_{-m}\times10^{-m}$$
$$=\sum_{i=-m}^{n-1}D_i\times10^i \qquad (2\text{-}1)$$

式中，D_i 是 D 的第 i 位的数码，可以是 0～9 十个符号中的任何一个，n 和 m 为正整数，n 表示小数点左边的位数，m 表示小数点右边的位数，10 为基数，10^i 称为十进制的权。

2．二进制数

二进制数由 0 和 1 两个符号组成，用符号 B 标识，遵循逢二进一的法则。一个二进制数 B 可用其权展开式表示为

$$\begin{aligned}(B)_2 &= B_{n-1}\times 2^{n-1}+B_{n-2}\times 2^{n-2}+\cdots+B_0\times 2^0+B_{-1}\times 2^{-1}+\cdots+B_{-m}\times 2^{-m}\\ &=\sum_{i=-m}^{n-1} B_i\times 2^i\end{aligned} \tag{2-2}$$

式中，B_i 为 1 或 0，2 为基数，2^i 为二进制的权，m、n 的含义与十进制表达式相同。二进制数通常用下标 2 表示。

例 2-1　二进制数 1011.01 可表示为

$$\begin{aligned}(1011.01)_2 &= 1\times 2^3+0\times 2^2+1\times 2^1+1\times 2^0+0\times 2^{-1}+1\times 2^{-2}\\ &=11.25\end{aligned}$$

3．十六进制数

十六进制数共有 16 个数字符号，0～9 及 A～F，用符号 H 标识，其计数规律为逢十六进一。一个十六进制数 H 的权展开式为

$$\begin{aligned}(H)_{16} &= H_{n-1}\times 16^{n-1}+H_{n-2}\times 16^{n-2}+\cdots+H_0\times 16^0+H^{-1}\times 16^{-1}+\cdots+H_{-m}\times 16^{-m}\\ &=\sum_{i=-m}^{n-1} H_i\times 16^i\end{aligned} \tag{2-3}$$

式中，H_i 为 0～F 范围取值，16 为基数，16^i 为十六进制数的权；m、n 的含意与上相同。十六进制数也可用下标 16 表示。

例 2-2　十六进制数 38EF.A4H 可表示为

$$\begin{aligned}(38EF.A4)_{16} &= 3\times 16^3+8\times 16^2+14\times 16^1+15\times 16^0+10\times 16^{-1}+4\times 16^{-2}\\ &=14\,575.640\,625\end{aligned}$$

4．其他进制数

除以上介绍的二、十和十六进制三种常用的进位计数制外，计算机中还可能用到八进制数。八进制数有 0～7 八个数符，用符号 O 标识，其计数规律为逢八进一。其权展开式可参照式（2-4）进行归纳。

下面给出任一进位制数的权展开式的一般形式。一般地，对任意一个 K 进制数 S，都可用权展开式表示为

$$\begin{aligned}(S)_k &= S_{n-1}\times K^{n-1}+S_{n-2}\times K^{n-2}+\cdots+S_0\times K^0+S_{-1}\times K^{-1}+\cdots+S_{-m}\times K^{-m}\\ &=\sum_{i=-m}^{n-1} S_i\times K^i\end{aligned} \tag{2-4}$$

式中，S_i 是 S 的第 i 位数码，可以是所选定的 K 个符号中的任何一个；n 和 m 的含义同上，K 为基数，K^i 称为 K 进制数的权。

需要注意的一点是，在默认情况下，十进制标识符 D 可省略，而其他进制数则需标明标识符，即当数字后无标识符时，计算机将默认其为十进制数。例如，1101B 是二进制数，而 1101 则默认为十进制数。

2.2.3　各种数制之间的转换

人类习惯的是十进制数，计算机采用的是二进制数，编写程序时为方便起见又多采用十六进制数，因此，必然会产生在不同计数制之间进行转换的问题。

1．非十进制数到十进制数的转换

非十进制数转换为十进制数的方法比较简单，只要将它们按相应的权表达式展开，再按十进制运算规则求和，即可得到它们对应的十进制数。

例 2-3　将二进制数 1101.101 转换为十进制数。

解　根据二进制数的权展开式，得

$$(1101.101)_2 = 1\times2^3+1\times2^2+0\times2^1+1\times2^0+1\times2^{-1}+0\times2^{-2}+1\times2^{-3}$$
$$= (13.625)_{10}$$

例 2-4　将十六进制数 64.CH 转换为十进制数。

解　根据十六进制数的权展开式，得

$$(64.C)_{16} = 6\times16^1+4\times16^0+C\times16^{-1} = 6\times16^1+4\times16^0+12\times16^{-1}$$
$$= (100.75)_{10}$$

2．十进制数转换为非十进制数

十进制转换为非十进制（*K* 进制）数时，整数和小数部分应分别进行转换。整数部分转换为 *K* 进制数时采用“除 *K* 取余”的方法，即连续除 *K* 并取余数作为结果，直至商为 0，得到的余数从低位到高位依次排列即得到转换后 K 进制数的整数部分；对小数部分，则用“乘 *K* 取整”的方法，即对小数部分连续用 *K* 乘，以最先得到的乘积的整数部分为最高位，直至达到所要求的精度或小数部分为零为止。

例 2-5　将十进制数 115.25 转换为对应的二进制数。

解

整数部分		小数部分
115/2=57……余数=1	（最低位）	0.25×2=0.5……整数=0（最高位）
57/2=28……余数=1		0.5×2=1.0……整数=1
28/2=14……余数=0		
14/2=7……余数=0		
7/2=3……余数=1		
3/2=1……余数=1		
1/2=0……余数=1		

从而得到转换结果$(115.25)_{10} = (1110011.01)_2$。

例 2-6　将十进制数 301.6875 转换为对应的十六进制数。

解

整数部分	小数部分
301/16=18……余数=D	0.6875×16=11.0000……整数=$(11)_{10}=(B)_{16}$
18/16=1……余数=2	
1/16=0……余数=1	

所以得 301.6875 = 12D.BH。

例 2-7 将十进制数 301.6875 转换为对应的八进制数。

解

整数部分	小数部分
301/8=37……余数=5	0.6875×8=5.5……整数=5（最高位）
37/8=4……余数=5	0.5×8=4.0……整数=4
4/8=0……余数=4	

所以得 301.6875 =$(455.54)_8$。

3．非十进制数之间的转换

由于 2^4=16，2^3=8，故二进制数与十六进制数、八进制数之间都存在有特殊的关系。一位十六进制数可用 4 位二进制数来表示，而一位八进制数可用 3 位二进制数表示，且它们之间的关系是唯一的。这就使得十六进制数与二进制数之间、八进制数与二进制数之间的转换都非常容易。由于二进制在书写上的麻烦和冗长，因此，在计算机应用中，虽然机器只能识别二进制数，但在数字的书写表达上更广泛地采用十六进制数或八进制。

计算机中常用的二进制数、十六进制数、八进制和十进制数之间的关系如表 2-1 所示。

表 2-1 数制对照表

十进制数	二进制数	十六进制数	八进制数
0	0000	0	0
1	0001	1	1
2	0010	2	2
3	0011	3	3
4	0100	4	4
5	0101	5	5
6	0110	6	6
7	0111	7	7
8	1000	8	10
9	1001	9	11
10	1010	A	12
11	1011	B	13
12	1100	C	14
13	1101	D	15
14	1110	E	16
15	1111	F	17

将二进制数转换为十六进制数的方法：从小数点开始分别向左和向右把整数和小数部分每 4 位分为一组。若整数最高位的一组不足 4 位，则在其左边补零；若小数最低位的一组不足 4 位，则在其右边补零。然后将每组二进制数用对应的十六进制数代替，则得到转换结果。

同样的方法，可实现二进制数到八进制数的转换，即从小数点开始分别向左和向右将

数每 3 位分为一组。若整数最高位的一组不足 3 位，则在其左边补零；若小数最低位的一组不足 3 位，则在其右边补零。

相应的，十六进制数和八进制数转换为二进制数时，可用 4 位/3 位二进制代码取代对应的一位十六进制/八进制数。

例 2-8　将二进制数 110100110.101101B 转换为十六进制数。

解

二进制数　　0001　1010　0110．1011　0100

↓　↓　↓　↓　↓

十六进制数　　1　A　6　．　B　4

所以得 $(110100110.101101)_2 = 1A6.B4H$。

例 2-9　将八进制数（273.64）$_8$ 转换为二进制数。

解

八进制数　　2　7　3　．　6　4

↓　↓　↓　↓　↓

二进制数　　010　111　011　．　110　100

从而得$(273.64)_8$= 010111011.110100B。

2.3　二进制数的表示和运算

2.3.1　二进制数的表示

在计算机中，用于表示数值大小的数据称为数值数据。计算机可以处理的数值可以是整数，也可以是小数。对整数的处理比较容易，但对小数，就会有些麻烦。例如，小数点处于不同的位置，其数的大小会不同。计算机中，对小数点位置的表示有定点表示法和浮点表示法两种。

1．数的定点表示

定点表示法就是小数点固定在一个规定的位置上。用这种方法表示的数称为定点数。

1）定点小数的表示

为方便起见，定点小数的表示通常都是把小数点固定在最高数据位的左边，变为纯小数。如果考虑数的符号，小数点的前边可以再设符号位。纯小数的定点小数可表示为

$$X=X_s .X_{-1}X_{-2}\cdots X_{-(n-1)}X_{-n}$$

这里，X_s 是符号位，用于表示数的正负。小数点是人为规定的，X_{-1} 至 X_{-n} 为数码部分，表示数值大小。其中，X_{-1} 表示最高有效位，X_{-n} 是最低有效位。当数码有 n 位时，定点小数所能表示的数值范围（又称为表数范围）为

$$-(1-2^{-n})\leqslant X\leqslant(1-2^{-n}) \qquad (2\text{-}5)$$

定点小数表示法在运算前，需要对原始数据事先进行处理，即需将用到的数先通过合适的“比例因子”转化为纯小数或纯整数，计算的结果又要再用比例因子折算成真实值。另外，这种方法能表示的数的范围小，精度也较低。

定点小数表示法比较节省硬件，主要用于早期计算机中。现代通用计算机都已能处理与计算多种类型的数值，定点小数表示方法目前主要用于表示浮点数的尾数部分。

2）整数的表示

纯小数是将小数点固定在最高有效位之前，若将小数点固定在最低有效位之后，则此时的数就变成了下边讨论的纯整数。任意一个整数都可表示为

$$X=X_sX_{n-1}\cdots X_1X_0 \qquad (2\text{-}6)$$

同样，X_s 表示符号，后边的 n 位表示数值部分。

对于用 n 位二进制码表示的带符号的二进制数，纯整数所能表示的数值范围为

$$-(2^{n-1})\leqslant X\leqslant 2^{n-1}-1$$

若不考虑符号，即没有符号位（也可以简单理解为都是正数），则 n 位二进制码表示的不带符号的二进制纯整数表数值范围为

$$0\leqslant X\leqslant 2^{n-1}-1$$

不带符号的二进制码可以被看作一串二进制位的某种组合，如一个字符的编码。

计算机往往使用几种不同的二进制位数来表示一个整数，如 8 位、16 位、32 位、64 位等。不同位数（字长）的整数的表数范围不同，占用的存储空间也不一样。

2．数的浮点表示法

浮点数，是指小数点的位置可以左右移动的数据。在十进制中，一个数可以写成多种表示形式，如 58.123，可以写成 $0.581\ 23\times10^2$，$0.058\ 123\times10^3$，$58\ 123\times10^{-3}$ 等。同样，一个二进制数也可以写成多种表示形式，如 1011.101 可以写成 0.1011101×2^4，0.01011101×2^5 等，即一个二进制数的表示形式为

$$X=\pm2^E\times F \qquad (2\text{-}7)$$

式中，E 称为阶码，即指数值，为带符号整数；F 表示尾数，通常是纯小数。将用阶码和尾数表示的数称为浮点数，将这种表示数的方法称为浮点表示法。

浮点数的一般格式如图 2-4 所示。

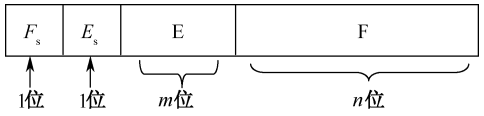


图 2-4　典型的浮点数格式

图 2-4 中，F_S 为尾符，表示尾数的符号，安排在最高位。它也是整个浮点数的符号位，表示该浮点数的正负；E_s 是阶符，表示阶码的符号，即指数的符号，决定符点数范围的大小；E 是阶码的值，F 是尾数的值。

可以看出，浮点数的表示不是唯一的。当小数点的位置改变时，阶码也随之相应改变，同一个数就可以有多种表现形式。为了便于浮点数之间的运算和比较，也为了提高数据的表示精度，规定浮点数的尾数用纯小数表示，即小数点右边第 1 位不为 0，阶码用整数表示。称这样的浮点数为规格化浮点数。对不满足要求的数，可通过修改阶码并同时左右移动小数点位置的方法使其变为规格化浮点数，这个过程又称为浮点数的规格化。

不论是浮点数还是定点数，在计算机中都是要存放在存储器中，而存储器的字长和容量都是有限的。因此，定点数有表数范围，浮点数同样也有。浮点数的表数范围主要由阶

码决定，精度则主要由尾数决定。采用图 2-4 格式所表示的规格化二进制浮点数的表数范围为

$$2^{-(1+2^m)} \leqslant |X| \leqslant (1-2^{-n}) \times 2^{2^m-1} \qquad (2\text{-}8)$$

早期计算机中只有定点数据表示，采用定点数的有优点是硬件结构比较简单，缺点除数据的表示范围比较小之外，主要是必须在运算前将所有参加运算的数据的小数点都对齐到最高位，运算结束后又要在恢复。使运算速度比较慢，也浪费很多存储空间。现在，随着硬件成本的大幅降低，现代通用计算机中都能够处理包括定点数、浮点数等在内的多种类型的数值。引入浮点数表示法，可使数的表示范围、精度，以及运算速度大幅提高。有关浮点数的进一步描述，请有兴趣的读者参阅其他相关书籍。

2.3.2　二进制数的算术运算

二进制数只有 0 和 1 两个数符，故其运算规则比十进制数要简单很多。

1．加法运算

二进制的加法运算遵循“逢二进一”法则，具体如下：

0+0=0　　0+1=1　　1+0=1　　1+1=0（有进位）

例 2-10　求两个二进制数 10110110B 和 01101100B 的和。

解

```
进　位　 1 11111000
被加数　  10110110
加　数+)  01101100
-----------------
        1 00100010
```

即 10110110B+01101100B=100100010B。

2．减法运算

对二进制数减法，同样有如下法则：

0−0=0　　1−0=1　　1−1=0　　0−1=1（有借位）

例 2-11　求两个二进制数 11000100B 和 00100101B 的差。

解

```
借　位　 01111110
被减数　  11000100
减　数-)  00100101
-----------------
          10011111
```

即 11000100B−00100101B=10011111B。

3．乘法运算

二进制数乘法与十进制乘法类似，不同的是因二进制数只由 0 和 1 构成，因此，其乘法更加简单。法则如下：

0×0=0　　0×1=0　　1×0=0　　1×1=1

即仅当两个 1 相乘时结果为 1，否则结果为 0。运算时若乘数位为 1，就将被乘数照抄

加于中间结果，若乘数位为 0，则加 0 于中间结果，只是在相加时要将每次中间结果的最后一位与相应的乘数位对齐。

例 2-12 求两个二进制数 1100B 与 1001B 的乘积。

```
    1 1 0 0          被 乘 数
  × 1 0 0 1          乘    数
-------------
    1 1 0 0          部 分 积
  0 0 0 0
0 0 0 0
1 1 0 0
-------------
1 1 0 1 1 0 0        乘    积
```

可得 1100B×1001B=1101100B。

从上述运算可以看出，从乘数的最低位算起，凡遇到 1，相当于在最终结果上加上一个被乘数，遇到 0 则不加；若乘数最低位是 1，被乘数直接加在结果的最右边；若次低位是 1，应左移一位后再相加；若再次低位是 1，应左移两位后再相加；……；以此类推。最后将移位和未移位的被乘数加在一起，就得到两数的乘积。这种将乘法运算转换为加法和移位运算的方法就是计算机中乘法运算的原理。

4．除法运算

二进制的除法是乘法的逆运算，其方法和十进制一样，而且比十进制除法更简单。

例 2-13 求两个二进制数 100111B 与 110B 的商。

```
        110.1
    ------------
110 ) 100111
      110
    ------
      0111
      110
    ------
       00110
       110
    ------
           0
```

二进制数的除法也是采用试商的方法求商数，分析上例的过程，可得出二进制的除法运算可转换为减法和右移运算。

2.3.3 机器数的表示和运算

不论是采用定点表示或是浮点表示法，计算机中存储和处理的二进制数可通称为机器数。十进制数有正数和负数，二进制数也可以有正负之分。十进制数的正数和负数分别用“+”和“-”表示，这是因为人能够识别“+”和“-”。由于计算机只能识别“0”和“1”，所以，机器数的正和负无法用“+”和“-”表示。

机器数分为无符号数和有符号数两种。无符号数，就是不考虑数的符号（这在十进制中没有对应），一个数中的每一位 0 或 1 都是有效的或有意义的数据，如 10010110B 是一个二进制数，该数中的每一位都是有意义的，由式（2-2）的权值表达式可以得出其对应的十进制数值为 150。

有符号数的含义：该数具有“正”或“负”的性质。此时，数据的最高位不是有意义的数据，而是符号位，用来表示数的正或负。与十进制数不同的是，机器数的正号和负号需要用 0 和 1 来表示，即在需要考虑数据符号的有符号数中，一个数的最高位的“0”或

“1”表示的是该数的性质，“0”表示正数，“1”表示负数，最高位不再是数据本身。以8位字长为例，D_7位是符号位，D_6～D_0为数值位；若字长为16位，则D_{15}为符号位，D_{14}～D_0为数值位。这样，有符号数中的有效数值就比相同字长的无符号数要小了，因为其最高位代表符号，而不再是有效的数据。

把符号位数值化了的数称为机器数，如00010101和100 1 0101就是机器数；而把原来的数值（数据本身）称为机器数的真值（也可以理解为绝对值），如+0010l01和−0010101。

机器数的表示方法有三种，即原码、反码和补码。

1．原码

原码表示法可以简单地表达成如下形式：

符号位+真值

一个数X的原码可记为$[X]_{原}$。在原码表示法中，不论数的正负，数值部分均为真值。

例2-14　已知真值X=+42，Y=−42，求$[X]_{原}$和$[Y]_{原}$。

解　因为$(+42)_{10}$=+0101010B，$(-42)_{10}$=−0101010B，根据原码表示法，得

$[X]_{原}$= 0 0101010　　$[Y]_{原}$ = 1 0101010

↑ ↑　　↑ ↑

符号位 数值部分　　符号位 数值部分

注意：在原码表示法中，真值0的原码可表示为两种不同的形式，即+0和−0。以8位字长数为例：

$$[+0]_{原} = 00000000$$

$$[-0]_{原} = 10000000$$

原码表示法的优点是简单易于理解，与真值间的转换较为方便；它的缺点是进行加减运算时较麻烦，不仅要考虑是做加法还是做减法，而且要考虑数的符号、绝对值大小及运算结果的符号，这使运算器的设计较为复杂，并降低了运算器的运算速度。

2．反码

反码是原码基础上的变形。对正数来讲，反码的表示方法与原码相同，即最高位为“0”，其余是数值部分。但负数的反码表示与原码不同，其最高位依然是符号位，用“1”表示，但其余的数值部分不再是原来的真值，而是将真值的各位按位取反，即原先为0的变为1，为1则变为0。

例2-15　已知真值X=+42，Y=−42，求$[X]_{反}$和$[Y]_{反}$。

解　因为$(+42)_{10}$=+0101010B，$(-24)_{10}$=−0101010B，根据反码表示法，得

$[X]_{反}$ = 00101010　　对正数：$[X]_{反}$ =$[X]_{原}$

$[Y]_{反}$ = 11010101　　对负数：$[Y]_{反}$ =$[Y]_{原}$的符号位不变，数值部分按位取反。

由该例可以看出，对一个用反码表示的负数，其数值部分不再是真值。

在反码表示法中，同原码一样，数值0也有两种表示形式（以8位字长数为例）：

$$[+0]_{反}=00000000$$

$$[-0]_{反}=11111111$$

在原码和反码表示法中，数值0的表示都不唯一，且运算器的设计比较复杂。因此，目前

在微处理器中已较少使用这两种表示方法（原码表示法主要用于浮点数中的阶码表示）。

3．补码

补码由反码演变而来，其定义：对正数，补码与反码和原码的表示方法相同，即最高位为“0”，其余是数值部分。但负数的补码表示与原码和反码不同，其最高位的符号位不变，但其余的数值部分是反码的数值部分加 1，即将原码的真值按位取反再加 1。

真值 X 的补码记为$[X]_{补}$。可用下式表述：

若 X≥0　　$[X]_{补}=[X]_{反}=[X]_{原}$

若 X＜0　　$[X]_{补}=[X]_{反}+1$

例 2-16　已知真值 X=+42，Y=−42，求$[X]_{补}$和$[Y]_{补}$。

解　因为 X＞0，所以：

$$[X]_{补}=[X]_{反}=[X]_{原}=00101010$$

因为 Y＜0，所以：

$$[Y]_{补}=[Y]_{反}+1=11010101+1=11010110$$

不同于原码和反码，数 0 的补码表示是唯一的。仍以 8 位字长数为例，由补码的定义知：

$$[+0]_{补}=[+0]_{反}=[+0]_{原}=00000000$$

$$[-0]_{补}=[-0]_{反}+1=11111111+1=\boxed{1}\,00000000$$

↓

自然丢失

即对 8 位字长来讲，最高位的进位因超出字长范围，会自然丢失，所以：

$$[+0]_{补}=[-0]_{补}=00000000$$

事实上，补码的概念在日常生活中也常见到，如钟表，若要从 9 点拨到 4 点，可以有两种拨法：

逆时针拨到 4 点　9−5=4

顺时针拨到 4 点　9+7=4

两个方向都能拨到 4 点，是因为在时钟系统中有 12 这个最大数，它称为该系统的模，它是自然丢失的。对时钟系统的模 12 而言，9−5=9+7，7 称为−5 的补数。所以，−5 的补数可用下式得到：

$$(-5)_{补}=12-5=7$$

即

$$9-5=9+(-5)=9+(12-5)=9+7=12+4=4$$

↓

模，自然丢失

由此可见，引入补码可以将减法运算转换为加法运算。可以将此概念推广到整个二进制系统。二进制计数系统的模 2^n，这里的 n 表示字长。

例 2-17　设字长 n=8，用补码的概念计算 96−20。

解　因为 n=8，故模为 $2^8=256$

则有 96−20=96+(−20)=96+(256−20)=96+236=256+76=76

↓

模

即在模为 2^n 的情况下，96−20=96+236

−20 的二进制表示为 11101100，该数正好是十进制的 236。这样，我们就利用了负数的补码概念，将减法运算转换成了加法运算。

利用补码实现加减运算的规则如下：

补码的加法规则：　$[X+Y]_{补}=[X]_{补}+[Y]_{补}$。

补码的减法规则：　$[X-Y]_{补}=[X]_{补}-[Y]_{补}=[X]_{补}+[-Y]_{补}$。

例 2-18　已知真值 X=+0110100B，Y=−1110100B，求$[X]_{补}+[Y]_{补}$=?

解　这里 X＞0，所以：

$[X]_{补}$=00110100B

Y＜0，所以：

$[Y]_{补}=[Y]_{反}+1$=10001011B+1=10001100B

由补码的加法运算规则知：

$[X+Y]_{补}=[X]_{补}+[Y]_{补}$=00110100B+10001100B=11000000B

例 2-19　设 X=+51，Y=+66，求$[X-Y]_{补}$＝？

由补码的减法运算规则：

$[X-Y]_{补}=[X]_{补}+[-Y]_{补}$

$X=(+51)_{10}=(+0110011)_2$，$[X]_{补}$=00110011B

$-Y=(-66)_{10}=(-1000010)_2$，$[-Y]_{补}$=10111110B

求$[X]_{补}+[-Y]_{补}$：

$$\begin{array}{r} 00110011 \\ +\ 10111110 \\ \hline 11110001 \end{array}$$

所以：

$[X-Y]_{补}$=11110001B

由补码运算规则知，两补码相加的结果为和的补码。以上两例的运算结果的符号位为 1，表示结果为负数。按照补码的定义，负数的补码=其原码按位取反+1，而原码的定义是符号位+数值。所以，当补码数的最高位为 1 时，表示该数是负数，即此时符号位后的数值“不是真的数值”，需要将其“还原”，即将数值部分按位取反加 1，得出真值。

所以，对例 2-19 的运算结果$[X-Y]_{补}$=11110001，符号位用“-”表示，数值部分按位取反加 1，就得到：X−Y=−0001111B=−15。

所以，计算机中引入补码的主要目的就是将减法运算转换为加法运算。另外，由上述分析已知，在补码表示法中，数 0 的表示是唯一的。因此，在微机中，凡涉及符号数都是用补码表示的。

2.4　逻辑运算与逻辑门

计算机由逻辑器件组成，要了解计算机的工作原理，首先需要了解逻辑运算的基本概念及基本逻辑门的符号表示。

2.4.1 逻辑运算

逻辑运算与算术运算不同，算术运算是将一个二进制数的所有位视为一个数值整体来考虑，低位的运算结果会影响到高位（如进位等）；而逻辑运算是按位进行的运算，例如，一个数最低位和另一个数的最低位运算，结果不会是对次低位产生影响，即逻辑运算没有进位或借位。基本逻辑运算包括“与”、“或”、“非”及“异或”四种运算。

1. “与”运算

“与”运算的规则是按位相“与”，一般用符号“∧”表示。其运算规则如下：

1∧1=1　　1∧0=0　　0∧1=0　　0∧0=0

即参加“与”操作的两位中只要有一位为0，则“与”的结果就为0，仅当两位均为1时，其结果才为1。相当于按位相乘（但不进位），又称为“逻辑乘”。

例2-20 计算11011010B∧10010110B=?

解

$$\begin{array}{r} 11011010 \\ \wedge\ 10010110 \\ \hline 10010010 \end{array}$$

即 11011010B∧10010110B=10010010B。

“与”运算是通过称为“与门”的逻辑器件实现的。“与门”可以有多位输入，但只有一位输出。仅当输入信号全为“1”时，输出为“1”；否则输出为“0”。

2. “或”运算

“或”运算是两个数按位相“或”的运算，又称为“逻辑加”，一般用符号“∨”表示。其规则如下：

0∨0=0　　0∨1=1　　1∨0=1　　1∨1=1

即参加“或”操作的两位中仅当两位均为0时，其结果才为0，只要有一位为1，则“或”的结果就为1。

试比较一下二进制数的“或”运算和加法运算的异同。

例2-21 计算11011001B∨10010110B=?

解

$$\begin{array}{r} 11011001 \\ \vee\ 11111111 \\ \hline 11111111 \end{array}$$

即 11011001B∨11111111B=11111111B。

同“与”运算类似，“或”运算通过称为“或门”的逻辑器件实现。“或门”也可以有多位输入，有一位输出。仅当输入信号全为“0”时，输出为“0”；否则输出为“1”。

3. “非”运算

“非”运算是按位取反的运算，即1的“非”为0，而0的“非”为1。“非”属于单边运算，即只有一个运算对象，其运算符为一条上横线。

$$\overline{1}=0 \qquad \overline{0}=1$$

例 2-22　求数 10011011 的非。

解　只要对 10011011 按位取反即可：

$$\overline{10011011B} = 01100100B$$

4．“异或”运算

“异或”运算相当“按位相加”（不进位），进行“异或”操作的两个二进制位不相同时，结果就为 l；两位相同时，结果为 0。“异或”运算符用符号⊕表示。

$0 \oplus 0 = 0$　　$1 \oplus 1 = 0$　　$0 \oplus 1 = 1$　　$1 \oplus 0 = 1$

例 2-23　计算 11010011B ⊕ 10100110B=?

解

$$\begin{array}{r} 11010011 \\ \oplus\ 10100110 \\ \hline 01110101 \end{array}$$

即 11010011B ⊕ 10100110B=01110101B。

二进制数的“异或”运算可以看作不进位的“按位加”，也可以看作不借位的“按位减”。同样，“非”运算和“异或”运算也有其相应的“逻辑门”。

2.4.2　基本逻辑门

上述各种逻辑运算都是通过“逻辑门”电路实现的。“逻辑门”是由若干半导体元件经过一定的组合构成的、能够实现某一种逻辑关系的集成电路。物理上的一片集成电路芯片上，通常会集成若干个具有同样逻辑关系的逻辑门，如将 4 个与门集成在一片芯片上构成的 4 个与门电路。

作为入门级教材，本书将不涉及逻辑门电路的内部结构，仅从应用的角度出发介绍几种计算机中的基本逻辑部件，包括它们的逻辑功能和符号表示。

1．与门（AND gate）

与门是对多个逻辑变量（取值范围只有“0”和“1”）进行“与”运算的门电路。对两个逻辑变量 A 和 B 的 “与”操作，其结果 Y 可表示为

$$Y = A \wedge B$$

它们的关系也可从表 2-2 所示的真值表中清楚地了解到，即仅在输入 A 和 B 均为 1 时，输出 Y 才为 1，A 和 B 中只要一个为 0，则 Y 就等于 0。以电路变化来表示，若采用正逻辑，则仅当与门的输入 A 和 B 都是高电平时，输出 Y 才是高电平，否则 Y 就输出低电平。

在电路连接上，两输入与门常用图 2-5 所示的两种逻辑符号之一来表示。

表 2-2　与门的真值表

A	B	Y
0	0	0
0	1	0
1	0	0
1	1	1

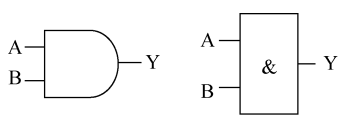


图 2-5　与门的逻辑符号

2．或门（OR gate）

或门是对多个逻辑变量进行“或”运算的门电路。对两个逻辑变量 A 和 B 的 “或”操作，它们的逻辑关系可用下边的运算表达式表示为

$$Y = A \vee B$$

即两个输入变量 A 和 B 中任意一个为 1，输出 Y 就为 1；仅当 A 和 B 都为 0 时 Y 才为 0。从电路的角度来说，当或门的输入 A 和 B 只要有一个是高电平，输出 Y 就为高电平，否则 Y 就输出低电平。

两输入或门常用图 2-6 所示的两种图符之一来表示，其真值表如表 2-3 所示。

表 2-3　或门的真值表

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	1

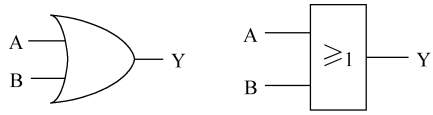


图 2-6　或门的逻辑符号

3．非门（NOT gate）

非门又称为反相器，是对单一逻辑变量进行“非”运算的门电路，其输入变量 A 与输出变量 Y 之间的关系可用下式表示为

$$Y = \overline{A}$$

非运算又称为求反运算，变量 A 上的上划线在数字电路中表示反相之意。非门的两种逻辑符号如图 2-7 所示，其真值表如表 2-4 所示。

表 2-4　非门的真值表

A	Y
0	1
1	0

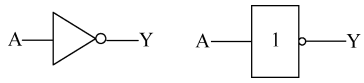


图 2-7　非门的逻辑符号

4．与非门（NAND gate）

“与”门与“非”门结合可以构成“与非门”。若输入变量为 A 和 B，则“与非”运算的过程是先对 A 和 B 进行“与”运算，再对结果进行“非”运算。可用下式表示为

$$Y = \overline{A \wedge B}$$

两输入与非门的两种逻辑符号如图 2-8 所示，图中逻辑符号图中的小圆圈表示“非”（本书将始终采用这种表示方法）。与非门的真值表如表 2-5 所示。

表 2-5　与非门的真值表

A	B	Y
0	0	1
0	1	1
1	0	1
1	1	0

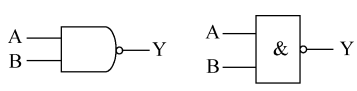


图 2-8　与非门的逻辑符号

5．或非门（NOR gate）

和与非门类似，或非门是“或”门与“非”门的结合，即先对输入 A 和 B 进行 “或”运算，再对其结果进行“非”运算，其运算表达式为

$$Y = \overline{A \vee B}$$

两输入或非门的两种逻辑符号如图 2-9 所示，真值表如表 2-6 所示。

表 2-6　或非门的真值表

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	0

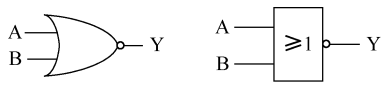


图 2-9　或非门的逻辑符号

逻辑门是构成计算机的最小单位。复杂的计算机就是由成千上万这样的逻辑门电路经过组合构成的。因此，基本逻辑门及其逻辑关系是学习计算机科学非常重要的基础。

习题

1．计算机系统主要由（　　）和（　　）组成。

2．说明以下计算机中的部件是属于主机系统、软件系统还是属于外部设备。

（1）CPU（　　）；

（2）内存条（　　）；

（3）网卡（　　）；

（4）键盘和鼠标（　　）；

（5）显示器（　　）；

（6）Windows 操作系统（　　）。

3．控制芯片组是主板的核心部件，它由（　　）部分和（　　）部分组成。

4．软件系统包括（　　）软件和（　　）软件。

5．在微机中，信息的最小单位是（　　）。

6．在计算机中，IB=（　　）bit，1KB 表示的二进制位数是（　　）位。

7．完成下列数制的转换：

（1）10100110B=（　　）D=（　　）H；

（2）0.11B =（　　）D；

（3）253.25 =（　　）B=（　　）H；

（4）1011011.101B=（　　）O=（　　）H=（　　）D。

8．完成下列二进制数的算术运算：

（1）10011010+01101110 =（　　）；

（2）11001100−100=（　　）；

（3）11001100×100=（　　）；

（4）11001100÷1000=（　　）。

9．写出下列真值对应的原码、反码和补码：

（1）X=−1110011B；

（2）X=−71D；

（3）X=+1001001B。

10．写出符号数 10110101B 的反码和补码。

11．已知 X 和 Y 的真值，求$[X+Y]_{补}$=？X+Y=？

（1）X=−1110111B，Y=+1011010B；

（2）X=56，Y=−21。

12．已知 X=−1101001B，Y=−1010110B，求$[X-Y]_{补}$=？X−Y=？

13．完成下列二进制数的逻辑运算：

（1）10110110∧11010110=（　　）；

（2）01011001B∨10010110=（　　）；

（3）$\overline{11010101}$=（　　）；

（4）11110111B⊕10001000=（　　）。

14．若“与门”的 3 位输入信号分别为 1、0、1，则该“与门”的输出信号状态为（　　）。若将这 3 位信号连接到或门，那么或门的输出又是什么状态？

15．在图 2-10 中，要使 Y=0，A1～A4 的状态必须为

（a）：（　　）；（b）：（　　）；

（c）：（　　）；（d）：（　　）。

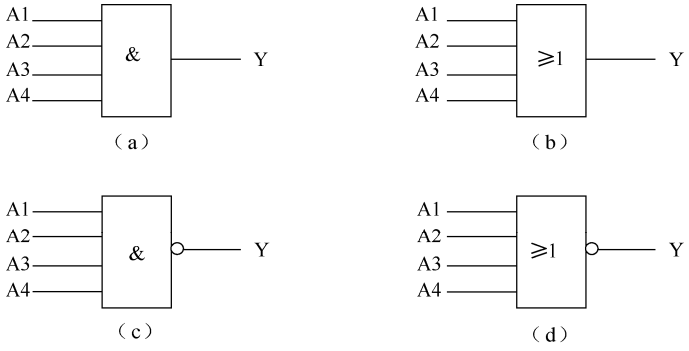


图 2-10　题 11 图

第 3 章　微型计算机系统

引言

也许很多读者没有考虑过我们今天已认为是理所当然的问题：为什么可以在屏幕上同时打开多个“窗口”？为什么我们总希望内存越大越好？我们把文件存到了硬盘的什么地方？是怎么放进去的？这一系列问题的答案，就隐藏在计算机硬件系统和操作系统工作原理中。

图灵奠定了计算机的理论基础，冯·诺依曼则创建了现代计算机的体系结构和基本原理。历经半个多世纪的发展，计算机的功能虽早已今非昔比，但其工作原理和体系结构在总体上依然是冯·诺依曼计算机。

本章通过对微型计算机硬件系统、微机基本工作原理、操作系统等三个知识模块的介绍，帮助读者了解计算机系统，揭晓上述问题的答案。

教学目的

- 理解微型计算机硬件系统的基本组成及其主机系统各主要部件的功能。
- 了解图灵机和计算机的关系。
- 深入理解冯·诺依曼计算机的基本结构和工作原理。
- 理解微型计算机的一般工作过程。
- 了解数据流计算机结构和哈佛结构。
- 了解操作系统的基本功能。
- 深入理解进程的基本状态及进程的生命周期。
- 理解操作系统中存储器管理的功能。
- 了解文件、文件的组织结构及文件管理的功能。

3.1　微型计算机硬件系统

微型计算机的硬件系统分为主机和外部设备两大部分。由于外部设备的多样性和复杂性，本书对硬件系统的介绍将只限于对主机部分的简要描述。

微型机的主机部分主要包括微处理器、存储器、总线及输入/输出接口等四个部分，如图 3-1 所示。

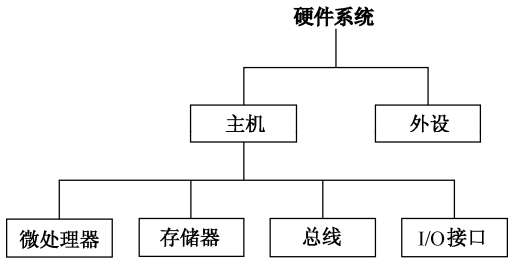


图 3-1　微机硬件系统概念结构

3.1.1　微处理器

微处理器（Microprocessor）又称为中央处理单元（Central Processing Unit，CPU）或微处理单元（Microprocessing Unit，MPU），是微型计算机的核心芯片，也是整个系统的运算和指挥控制中心。不同型号的微型计算机，其性能的差别首先在于其 CPU 性能的不同，而 CPU 性能又与它的内部结构有关。无论哪种 CPU，其内部的基本组成都大同小异，都主要包括控制器、运算器和寄存器组三个部分。图 3-2 给出了 CPU 的基本结构简图。

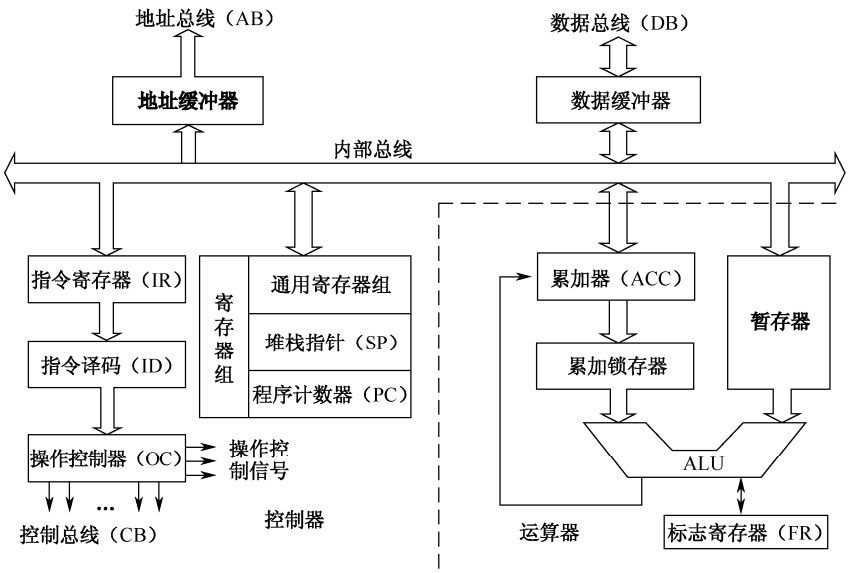


图 3-2　微处理器基本结构示意图

运算器的核心部件是算术逻辑单元 ALU（Arithmetic and Logic Unit），顾名思义，ALU 的主要功能就是在控制信号的作用下可完成加、减、乘、除等算术运算、移位操作及各种逻辑运算（扩大一点讲就是执行指令）。现代新型 CPU 的运算器还可完成各种浮点运算。运算可能产生的中间结果可以存放在累加器和暂存器中（它们也是寄存器），标志寄存器（FR）中存放的是运算结果的特征。例如，是否有进位、结果是否为零、结果的符号位状态等。

控制器主要用于产生控制和协调整个 CPU 工作所需要的时序逻辑，并负责完成与内存和输入/输出接口间的信息交换，如读取指令、读取参加运算的数据、存放指令执行的结果等。读取程序指令的地址由程序计数器（Program counter，PC）产生。PC 又称为指令指针，

它表示下一条要读取的指令在内存中的存放地址；读取或写入数据的地址则由指令本身确定（即图 3-12 所示指令格式中的操作数部分）。这些地址信号通过地址总线传送。

微处理器的工作基准是时钟信号（就像人类的作息基准是时间一样），这是一组周期恒定的连续脉冲信号。CPU 在不同的时刻执行不同的操作，这些操作在时间上有着严格的关系，这就是时序。时序信号由控制器产生，控制微处理器的各个部件按照一定的时间关系有条不紊地完成要求的操作。CPU 执行一条指令所需要的时钟个数不是固定的，有些指令仅需一个时钟周期即可完成，有些指令可能需要多个时钟周期才能完成。

衡量一个微处理器性能的高低，最重要的是执行指令（或程序）所用时间的多少。而所用时间的多少又与时钟速度和执行一条指令所需的时钟脉冲个数有关。微处理器的时钟速度越快、执行指令需要的时钟脉冲个数越少，指令执行的速度就越快。这也是为什么在同等情况下，CPU 的钟频越高，运算速度越快的原因。

程序员编写完成的程序，首先是存放在外存储器（如硬磁盘）中，在被执行前则由操作系统调入内存。执行时由控制器负责从内存储器中依次取出程序的各条指令，并根据指令的要求，向微机的各个部件发出相应的控制信号，使各部件协调工作，从而实现对整个微机系统的控制。

控制器一般由程序计数器、指令寄存器和操作控制电路组成，是整个 CPU 的指挥控制中心，对协调整个微型计算机有序工作极为重要。

寄存器组是 CPU 内部的若干个用于暂时存放数据的存储单元。包括多个专用寄存器和若干通用寄存器。专用寄存器的作用是固定的，如程序计数器用于指示下一条要取指令的地址；堆栈指针用于标示堆栈的栈顶位置[1]；标志寄存器用于存放当前运算结果的特征（如有无进位，结果是否为零，运算有无溢出）等。通常寄存器可由程序员规定其用途，其数目因 CPU 而异，如第三代微处理器 8086 CPU 中有 8 个 16 位通用寄存器，而 Pentium 4 中则有 8 个 32 位通用寄存器、8 个 80 位浮点数据寄存器、8 个支持单指令多数据操作的 64 位寄存器等可供程序员使用。由于有了这些寄存器，在需要重复使用某些操作数或中间结果时，就可将它们暂时存放在寄存器中，避免对存储器的频繁访问，从而缩短指令长度和指令执行时间，同时也给编程带来很大的方便。

除了程序员可用的寄存器外，微处理器中还有一些不能直接为程序员所用的寄存器，如累加锁存器、暂存器和指令寄存器等，它们仅受内部定时与控制逻辑的控制。

数据或指令在 CPU 中的传送通道称为 CPU 内部总线（Bus）。例如，运算所需的操作数、运算结果及指令码都是通过数据总线传送的。

在现代计算机中，指令的执行都采用了并行流水线方式。因此，实际的微处理器结构要比图 3-2 所示复杂很多，功能也强很多，不仅包含了这些基本的部件，还包括有专门的存储器管理、指令和数据缓存、流水线管理和执行部件等。有兴趣的读者可进一步学习有关微型计算机原理等相关课程。

3.1.2 存储器

存储器（Memory）的功能就是存放各种数据。这里的数据是广义的，包括数值、文本

[1] 堆栈是内存中的一个特殊的区域，用于存放暂时不用又必须保存的信息。它相当于日常生活中的储物箱，最先放进去的在最底下（栈底），后放入的在上。栈顶标示着堆栈中的信息容量。若栈顶与栈底重合，则表示栈为空。

及各类多媒体信息。对存储器的操作有两种，即“读”和“写”。“读”表示从存储器中输出数据，又称为读取；“写”表示向存储器输入数据，又称为写入。对存储器的读写，可以按字节、字或块。

计算机中的存储器从大的类型上可以分为内存和外存两类。内存由半导体材料制成，属于主机部分（对应着 2.1.2 节中讲到的内存条），用于存放计算机当前运行的程序（包括确保计算机运行所必须的程序）及运算的数据。内存需要后备电源，当断电时，其上存放的信息将丢失。

外存包括联机外存和脱机外存两种，脱机外存有光驱、磁带、移动存储器等，由复合材料（如光盘）、磁性材料或半导体材料（如优盘）构成。它们可以脱离计算机而存在，所以理论上可以存放无限多的数据。外存中存放着处理器不直接执行，但可以长期保留的数据。

1．内存

内存储器（又称为半导体存储器）主要用于存放数据（包括原始数据、中间结果和最终结果）和当前执行的程序，与外存储器相比，其主要的特点是可以与 CPU 直接进行信息交换。另外，相对于外存储器，内存还具有存取速度快、容量小、单位字节容量价格较高等特点。按照工作方式的不同，内存储器又可分为随机存取存储器（Random Access Memory，RAM）和只读存储器（Read Only Memory，ROM）两类。RAM 是微机中主内存的主要构成部件，其主要特点是可以随机进行读取和写入操作，但掉电后信息会丢失。

如同一栋大楼是由若干个房间组成一样，内存由若干个单元组成，如图 3-3 所示。大楼中的每个房间都有门牌号码，且每个号码在楼内都是唯一的，目的是便于寻找；同样，内存中的每个单元也有“门牌号码”，称为地址码，每个单元的地址在内存中也是唯一的。由于计算机只能识别二进制，所以，内存中的地址码都是用二进制表示的。地址码的长度依内存单元的个数（称为容量）而定。例如，4 个单元的内存的地址码只需要 2 位二进制的就可以表示（明白为什么吗？），而 4G 个单元的内存中，每个单元的地址则需要 32 位二进制码来表示。

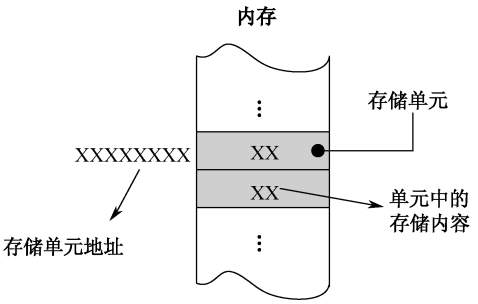


图 3-3　内存结构示意图

在微机系统中，内存的每个单元都存放 8 位二进制码，即 1B 数据。内存的容量就是指它具有的单元数，如常说的 2GB 内存，意思就是该内存有 2G（2^{30}）个单元，每单元中有 1 字节数据。

对内存的读/写操作通常按“字”进行，不同的系统“字”的长度不同。目前的微型机多为 64 位机，其在一个周期中能够对内存读出或写入 8B 数据。

2．硬盘

硬磁盘是微机中主要且必备的存储部件，由多片磁性材料制造的盘片叠加在一起构成，如图 3-4 所示。每个盘片有两个记录面（每个记录面对应一个磁头，所以，也用磁头数表示记录面数），每个记录面上是一系列称为磁道的同心圆（多个记录面上的同心圆叠放在一

起就构成柱面），每个磁道又被划分为若干个扇区（Sector）。硬磁盘就是按记录面、磁道和扇区来对数据进行组织的。对硬磁盘进行读/写操作的基本单位是扇区，每个扇区的容量为512B，而整块硬盘的容量则为

磁头数×柱面数×扇区数×512B　　（3-1）

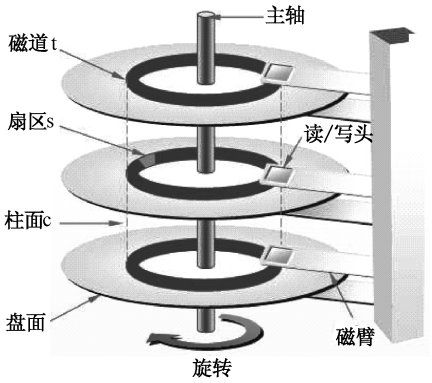


图 3-4　硬磁盘结构示意图

例 3-1　设已知磁头数为 16，柱面数为 4096，扇区数为 64，求该硬盘的容量。

由式 3-1 可得

该硬盘的容量=16×4096×64×512＝2 147 483 648B=2GB

外存储器的主要作用是保存各种希望由计算机保存和处理的信息。相对于内存，具有存储容量大（目前，微机硬磁盘的常规配置为 1TB，而内存通常为 4GB 或 8GB）、速度慢、单位字节容量价格低、不能与处理器直接进行信息交换等特点。外存储器虽然也安装在主机箱中，但属于外部设备的范畴。

外存与处理器之间的信息交换需要通过输入/输出接口。常用硬盘接口标准有 ATA（Advanced Technology Attachment，又称 IDE 或 EIDE）、SCSI（Small Computer System Interface）、SATA（Serial ATA）等，它们定义了外存储器（如硬盘、光盘等）与主机的物理接口。目前，最为流行的是使用 SATA 接口的硬盘，又称为串口硬盘。

串口硬盘采用串行方式传输数据，一次传送 1 位数据。相对于并行 ATA 来说，其数据传输率较高（Serial ATA 2.0 的数据传输率达 300MB/s，远高于 ATA 的 133MB/s 的最高数据传输率），同时接口的针脚数目很少（仅用 4 支针脚，分别用于连接电缆、连接地线、发送数据和接收数据），使连接电缆数目变少，降低了系统能耗和减小系统复杂性。

想一想：为什么外存储器属于外部设备的范畴？

3．存储器系统

虽然存储器按制造材料、存取速度、单位容量价格等方面可以分为上述两大类，但随着计算机技术的发展，存储器的地位不断提升，以运算器为核心的系统结构在逐渐转变为以存储器为核心，它不仅要求每一类存储器能够具有更高的性能，而且希望通过硬件、软件或软硬件结合的方式将不同类型的存储器组合在一起，从而获得更高的性价比，这就是存储器系统。现代微型计算机中的存储器，确切地讲是指存储器系统，它和存储器可是两个不同的概念。

常见的存储系统有两类，一类是由主内存和高速缓冲存储器（Cache）构成的 Cache 存储系统，另一种是由主存和磁盘存储器构成的虚拟存储系统。前者的主要目标是提高存储器系统的存取速度，而后者则主要是为了增加系统的存储容量。

1）Cache 存储器系统

Cache 由高速静态存储器（SRAM）组成，存取速度较普通主存快，周期一般小于 1ns（事实上，由于 Cache 大都与 CPU 集成在一起，其工作速度与 CPU 同步）。Cache 在系统中的位置可用图 3-5 来示意，其中存放的数据是主存中某一块（块大小与 Cache 相当）数据的映像（备份）。Cache 的主要作用如下：当 CPU 要访问（读或写）内存时，首先访问 Cache，若成功（命中）则继续；若访问不成功，再访问主存。

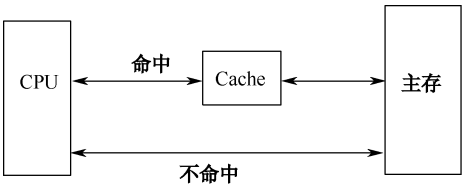


图 3-5　Cache 存储器系统示意图

Cache 存储系统由硬件系统管理，其设计目标是保证在一定的程序执行时间内，CPU 需要的数据和代码大都能在 Cache 中访问到（较高的命中率）。由于 Cache 的速度远高于主存的速度，容量远小于主存的容量。所以，当 Cache 的访问命中率较高时，从整个 Cache 存储器系统的角度看，其存取速度与 Cache 的速度接近，而容量是主内存的容量，且由于 Cache 容量较小，因此，系统单位容量价格与主存接近。

2）虚拟存储器系统

虚拟存储器系统由主内存和部分硬磁盘组成（图 3-6），由操作系统负责管理。其主要设计思想是希望提供一个比实际内存空间大得多的地址空间（即虚拟存储空间）。由于 CPU 执行程序时，其操作的对象（包括程序本身及其运算的数据）都必须要在内存中（放在外存中可就来不及了），因此，内存的大小就变得非常重要。早期的程序员在设计程序时，需要考虑其编写的程序是否可以一起放进内存。有了虚拟存储器系统，程序员编写程序时就不必再考虑内存容量的大小了，反正有足够大的虚拟存储空间可用。事实上，今天计算机中的程序在运行时，应用程序员根本不知道（也没必要知道）自己所编写的程序什么时候进入了内存及放进了内存的什么地方。主内存与磁盘间的数据交换都由操作系统全权负责了，只有系统程序员会知道中间的过程。

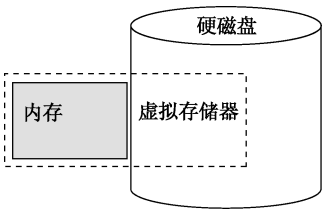


图 3-6　虚拟存储器示意图

虚拟存储器系统工作原理与 Cache 存储器系统类似。

3.1.3　总线

总线（Bus）是一组信号线的集合，是计算机系统各部件之间传输地址、数据和控制信息的公共通路。从物理结构来看，它由一组导线和相关的控制、驱动电路组成。在微型计算机系统中，总线常被作为一个独立部件看待。

微型计算机从诞生起就采用了总线结构。处理器通过总线实现与内存、外设之间的数据交换。计算机中最初的总线结构如图 3-7 所示，这也是“总线”最原始的含义，即在一组信号线上“挂接”多个部件（设备），这些部件分时共用这一组信号通道，任一时刻仅有一个部件能够利用该信道发送信息。

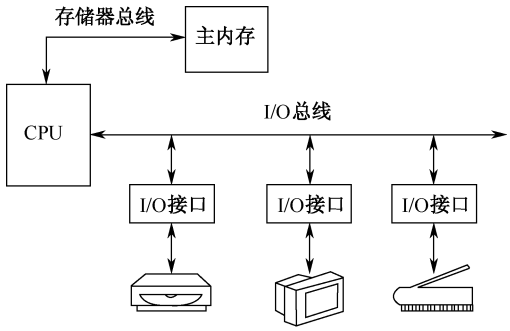


图 3-7　早期计算机中的总线结构

总线从传输信息的角度可分为三种类型：一是用于传输数据信息（计算机运行和处理的所有对象）的数据总线（Data Bus，DB）；二是用于传输地址信息（运算对象或运算结果在内存或接口中的存放处）的地址总线（Address Bus，AB）；三是用于传输控制信息（系统运行所需要的各种控制信号）的控制总线（Control Bus，CB）。在图 3-7 中，地址信息和多数控制信息由 CPU 发出，数据信息可以由 CPU 发送到内存或输入/输出接口（数据写入），也可以由内存或接口发送到 CPU（数据读取）。

图 3-7 的单总线结构存在一些缺陷。首先是所有部件都挂接在一条总线上，容易造成总线争用和拥堵；另外，由于总线上连接的部件在运行速度上存在差异，使总线的传输速度难以提高，使整个系统效率降低。

现代微机系统中的总线属于多总线结构，如图 3-8 所示。在这种结构中，总线除按传输信息的种类依然可以分为 DB、AB、CB 等三种类型外，还可以从层次结构上分为 CPU 总线（前端总线）、系统总线和外设总线。

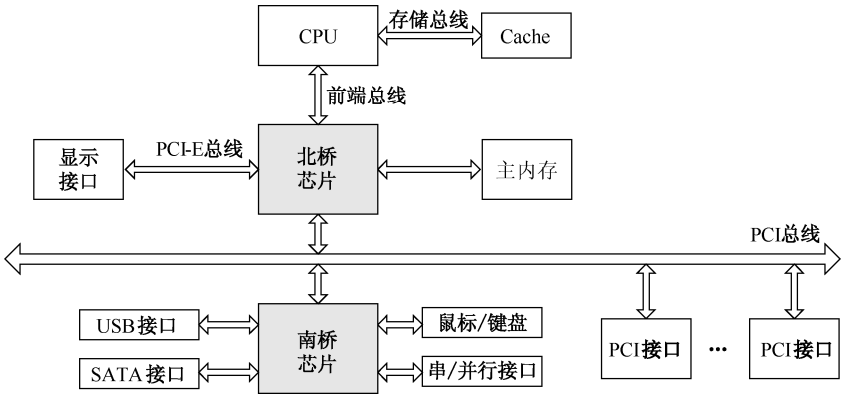


图 3-8　现代微机中的多总线结构

前端总线包括地址总线、数据总线和控制总线。一般是指从 CPU 引脚上引出的连接线，用来实现 CPU 与主存储器、CPU 与 I/O 接口芯片、CPU 与控制芯片组等芯片之间的信息传输，也用于系统中多个 CPU 之间的连接。前端总线是生产厂家针对其具体的处理器设计的，与具体的处理器有直接的关系，没有统一的标准。

系统总线又称为 I/O 通道总线，同样包括地址总线、数据总线和控制总线，是主机系统与外围设备之间的通信通道。在主板上，系统总线表现为与 I/O 扩展插槽引线连接的一组逻辑电路和导线。I/O 插槽上可插入各种扩展板卡，它们作为各种外部设备的适配器与外

设相连。为使各种接口卡能够在各种系统中实现“即插即用”，系统总线的设计要求与具体的 CPU 型号无关，而有自己统一的标准，各种外设适配卡可以按照这些标准进行设计。目前，常见的总线标准有 PCI 总线、PCI-E 总线等。

- PCI（Peripheral Component Interconnect）总线是外设互连总线的简称，是由美国 Intel 公司推出的 32/64 位标准总线，适用于 Pentium 以上的微型计算机，是目前微型机中应用最广泛的系统总线标准。
- AGP（Accelerated Graphics Port）总线，亦即加速图形端口。它是一种专为提高视频带宽而设计的总线规范。其视频数据的传输速率可以从 PCI 的 133MB/s 提高到 266MB/s（×1 模式——每个时钟周期传送一次数据）、533MB/s（×2 模式）、1.064GB/s（×4 模式）和 2.128GB/s（×8 模式）。严格地说，AGP 不能称为总线，因为它是点对点连接，即在控制芯片和 AGP 显示接口之间建立一个直接的通路，使三维图形数据不通过 PCI 总线，而直接送入显示子系统。这样就能突破由 PCI 总线形成的系统瓶颈。
- PCI-E（PCI Express）总线，它是目前最新的系统总线标准。虽然是在 PCI 总线的基础上发展起来的，但它与并行体系的 PCI 没有任何相似之处。它采用串行方式传输数据，依靠高频率来获得高性能，因此，PCI Express 也一度被人称为“串行 PCI”。

外设总线是指计算机主机与外部设备接口的总线，实际上是一种外设接口标准。目前，在微机系统中最常用的外设接口标准就是 USB（Universal Serial Bus，通用串行总线），可以用来连接多种外部设备。

3.1.4 输入/输出接口

没有输入/输出设备的计算机可以说是没有意义的。我们可以设想，如果没有显示器、没有鼠标和键盘，而仅有一台主机，可以做什么？

所有能够与计算机通信的设备都可以称为输入/输出设备，它们虽然千差万别，种类繁多，结构、原理各异，但都有一个共同的特点：都无法与系统直接进行信息交换。它们要想接收来自 CPU 的信息或将信息送入 CPU 去处理，必须通过一个中间环节，就是输入/输出接口（Input/Output Interface，I/O 接口），又称为 I/O 适配器（I/O Adapter）。

1．I/O 接口的功能

I/O 接口是将外设连接到系统总线上的一组逻辑电路的总称，也称为外设接口。其在系统中的作用如图 3-9 所示。在一个实际的计算机控制系统中，CPU 与外部设备之间常需要进行频繁的信息交换，包括数据的输入输出、外部设备状态信息的读取及控制命令的传送等，这些都是通过接口来实现的。由 I/O 接口在系统中的位置，使得接口电路应解决如下问题，也是接口应具有的功能：

图 3-9　I/O 接口在系统中的作用示意图

- **CPU 与外设的速度匹配**。CPU 与外设之间的工作时序和速度差异很大，要使两者之间能够正确进行数据传送，需要接口做“适配”。接口电路应具有信息缓冲能力，不仅应缓存 CPU 送给外设的信息，也要缓存外设送给 CPU 的信息，以实现 CPU 与外设之间信息交换的同步。

- **信息的输入/输出**。通过 I/O 接口，CPU 可以从外部设备输入各种信息，也可将处理结果输出到外设。同时，为保证数据传输的正确性，需要有一定的监测、管理、驱动等能力。
- **信息的转换**。外部设备种类繁多，其信号类型、电平形式等与 CPU 都可能存在差异。I/O 接口应具有信息格式变换、电平转换、码制转换、传送管理，以及联络控制等功能。
- **总线隔离**。为防止干扰，I/O 接口还应具备一定的信号隔离作用，使各种干扰信号不影响 CPU 的工作。

2．数据的输入/输出过程

系统从外设获取信息（读取数据），称为数据输入；反之将信息发送到外设则称为数据输出。CPU 与外部设备之间进行的信息交换实际上也就是数据的输入或输出。通过 I/O 接口进行数据输入和输出的过程可以简述为以下几个方面。

（1）当 CPU 要从外部设备读取数据时：

- 选择要访问的输入设备。CPU 将连接该外设的接口的地址放在地址总线上，标示出将要选择的设备。
- CPU 等候输入设备的数据成为有效。由于外设的速度相对 CPU 较低，因此，对数据的准备需要时间。这种“等候”将使 CPU 效率降低，但如果这个“等待”的工作由 I/O 接口来做，那么 CPU 在这段时间就可以执行其他的程序。所以，当 CPU 需要从外设读取数据时，首先由 I/O 控制器控制数据发送到接口的缓存中，再“通知”CPU。
- CPU 从数据总线读入数据，并放在一个相应的寄存器中。当数据进入接口缓存后，CPU 通过数据总线从接口中读取，这样所需要的时间就比较短了。

（2）当 CPU 要向外部设备输出数据时：

- CPU 把将连接该外设的接口的地址放在地址总线上，选择输出设备。
- CPU 把数据放在数据总线上，并用很短的时间将该数据送入接口。
- 接口收到数据后，“通知”所连接的外设。
- 输出设备将数据取走。

3．CPU 与外部设备的数据传输控制

由于不同的输入/输出设备本身工作速度差异很大，因此，对不同速度的外部设备，为确保数据传输时间上的同步，需要有不同的数据传输控制方法。

1）对极低速或简单外部设备

低速或简单外部设备，如开关、发光二极管、七段数码管等，属于“随时准备好”的外设，即任何时候都有确定的状态能够被输入或能接收 CPU 的输出。如对开关，因为其动作相对 CPU 的速度非常慢，CPU 可以认为输入的数据一直有效；对发光二极管或七段数码管，CPU 可以随时输出数据使其发光，因为对方始终处于可接收数据状态。所以，对这类设备，CPU 只要接收或发送数据就可以了。

2）对低速或中速的外部设备

由于这类设备运行速度和 CPU 不在一个数量级，或设备（如键盘）本身是在不规则时间间隔下操作。因此，CPU 与这类设备之间的数据交换通常采用异步传输方式。其工作过

程：当 CPU 要从外设接收一个字的数据时，首先查询外设的状态，如果状态表示该外设处于“准备就绪”，则 CPU 就从总线上接收数据，并在接收完数据后发出输入响应信号，告诉外设已将数据总线上的数据取走。外设收到响应信号后，将“准备就绪”状态标志复位，并准备下一个字的交换。

如果在 CPU 查询时外设没有“准备就绪”，则它会给出“忙”的标志，CPU 就进入循环等待，并在每次循环中询问外设的状态，一直到外设发出“准备就绪”信号后，才从外设接收数据。

CPU 向外设发送数据的过程与上述相似。外设先发出请求输出信号，之后 CPU 询问外设是否准备就绪。如果准备就绪，CPU 便发出准备就绪信号，并送出数据。外设接收数据以后，将向 CPU 发出“数据已经取走”的应答信号。

上述这种输入/输出方法常称为“应答式数据交换方式”或“查询工作方式”，其工作流程图如图 3-10 所示。

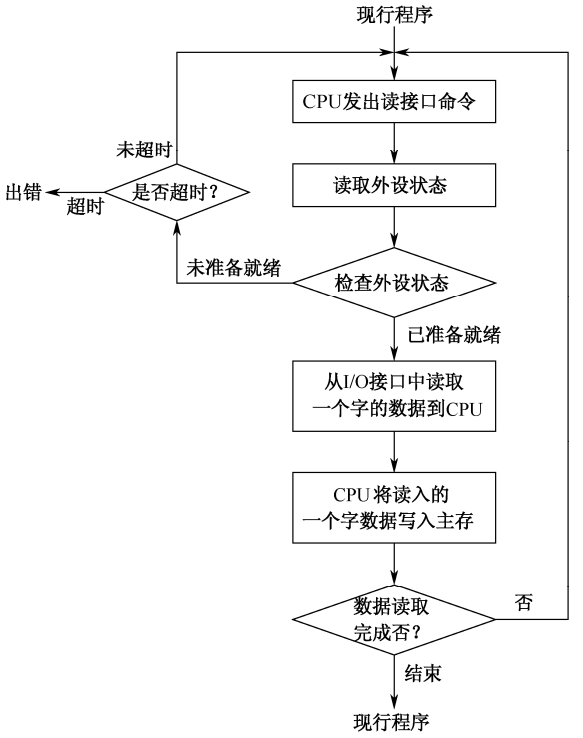


图 3-10　应答式数据交换工作流程

3）对高速外部设备

由于这类外设运算速度较快，且是以相等的时间间隔操作，而 CPU 也是以等间隔的速率执行输入/输出指令。因此，对这类设备的数据传送采用同步工作方式。一旦 CPU 和外设发生同步，它们之间的数据交换便靠时钟脉冲控制来进行。

3.2　冯·诺依曼计算机

计算机历经半个多世纪的发展，在运算速度、存储能力及整体功能上都有了巨大的进

步，根据莫尔定律[1]，计算机的整体性能每两年就会翻一番，事实也的确如此。今天，一台普通个人计算机的性能已远远超过 50 年前的巨型计算机。虽然发生了如此大的进步，但如今微型计算机的体系结构和工作原理依然沿用着冯・诺依曼 50 多年前的设计思想。

3.2.1　程序和指令

现代计算机不仅能够进行各种复杂的数值计算，还能够模拟人类的思维进行分析和处理各种事务。那么我们是否考虑过，计算机为什么能够做这样多的事情？它是如何完成每一项任务的？

计算机之所以能够按照要求完成一项一项的工作，是因为人向它发出了一系列的“命令”，这些命令通过输入设备以一定的方式送入计算机，并且能够为计算机所识别。我们将这种能够被计算机识别的命令称为指令，一台计算机能够识别的所有指令的集合称为该计算机的指令系统，而保证对指令的这种执行能力的是计算机的硬件系统。

当人们需要计算机完成某项任务的时候，首先要将任务分解为若干个基本操作的集合，并将每一种操作转换为相应的指令，按一定的顺序组织起来，这就是程序。计算机完成的任何任务都是通过执行程序完成的。例如，在需要解一道数学题时，要先把题目的解算步骤按照一定的顺序用计算机能够识别的指令书写出来，命令计算机执行规定的操作。这些指令的序列就组成了程序，如图 3-11 所示。

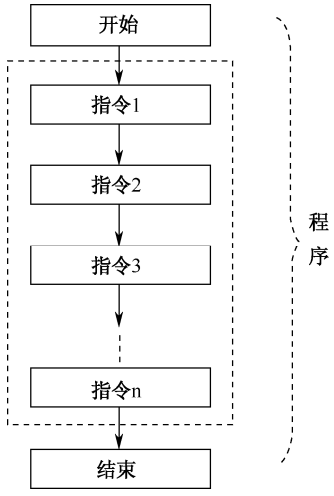


图 3-11　程序中的指令

计算机硬件能够直接识别并执行的指令称为机器指令。它们全部由“0”和“1”这样的二进制编码组成，其操作通过硬件逻辑电路实现。

不同的计算机系统通常都具有自己特有的指令系统，其指令在格式上也会有一些区别，但一般都包含这样三种信息，即完成何种操作（操作性质，如加、减、乘、除等）、对谁操作（操作的对象），以及操作结果的存放处。表征指令操作性质或者功能的称为操作码，表征操作对象的称为操作数（或地址码[2]）。指令的一般格式如图 3-12 所示。

每台计算机都拥有由各种类型的机器指令组成的指令系统。指令系统的功能是否强大、指令类型是否丰富，决定了计算机的能力，也影响着计算机的结构。指令的不同组合方式，可以构成完成不同任务的程序。计算机严格按照程序安排的指令顺序，有条不紊地执行规定的操作，完成预定任务。因此，程序是实现既定任务的指令序列，其中的每条指令都表示计算机执行的一项基本操作。一台计算机的指令种类是有限的，但通过人们的精心设计，可编写出无限多个实现各种任务处理的程序。

操作码（OPC）	操作数的目标地址，操作数的源地址

图 3-12　指令格式

[1] 集成电路芯片中的晶体管数量每隔 24 个月会翻一番。
[2] 表示运算数据及运算结果的存放处。

3.2.2 冯·诺依曼计算机基本结构

“冯·诺依曼计算机”的核心设计思想主要体现为存储程序控制原理、以运算器为核心、采用二进制。可进一步描述如下：

- 将计算过程描述为由许多条指令按一定顺序组成的程序，并放入存储器保存。
- 程序中的指令和数据都采用二进制编码（抛弃了十进制计数的设计思路），且能够被执行该程序的计算机所识别。
- 指令和数据可一起存放在存储器中，并作同样处理。
- 指令按其在存储器中存放的顺序执行，存储器的字长固定并按顺序线性编址。
- 由控制器控制整个程序和数据的存取及程序的执行。
- 计算机有运算器、逻辑控制装置、存储器、输入和输出设备五个部分组成，以运算器为核心，所有的执行都经过运算器。

冯·诺依曼计算机的设计思想简化了计算机的结构，大大提高了计算机的工作速度。图 3-13 是冯·诺依曼计算机的结构示意图。

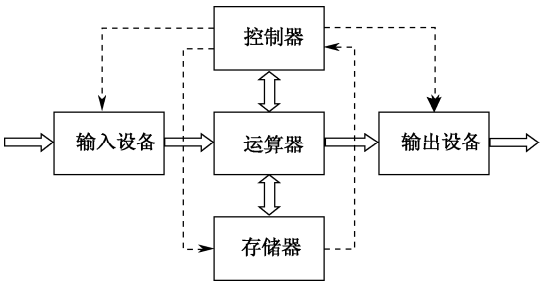


图 3-13 冯·诺依曼计算机结构示意图

半个多世纪过去，虽然计算机软硬件技术都有了飞速的发展，但直至今天，微型计算机的基本结构形式并没有明显的突破，仍属于冯·诺依曼架构。计算机的基本工作原理仍然是存储程序控制原理。当然，二进制也依然是计算机硬件唯一能够直接识别的数制。

3.3 微型计算机的基本工作原理

计算机的工作过程就是执行程序的过程，而程序是指令的序列。所以，计算机的工作过程就是一条条执行指令的过程。

3.3.1 指令的执行过程

由 3.2.1 节知，指令是控制计算机完成某种操作，并能够被计算机硬件所识别的命令。因此，指令才有如图 3-12 所示的格式，即包含了指令码和操作数。根据冯·诺依曼计算机的原理，程序在被执行前先要存放在（内）存储器中（为什么？学完 3.5 节就应该清楚了），而程序的执行需要由 CPU 完成。因此，计算机在执行程序时，首先需要按某种顺序将指令从内存储器中取（一次读取一条指令）出并送入微处理器，处理器分析指令要完成的动作，明确其操作性质和操作的对象，再去存储器中读取相应的操作数（如果需要的话），然后执

行相应的操作，最后将运算结果存放到内存储器中（如果这个结果不需要送到内存当然就可以不送了）。这一过程直到遇到结束程序运行的指令才停止。

因此，指令的执行过程可简单地描述为五个基本步骤：取指令、分析指令、读取操作数、执行指令和存放结果。图 3-14 给出了一条指令的执行流程。

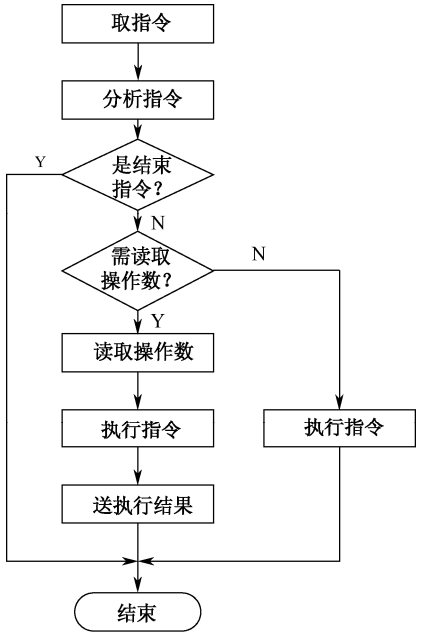


图 3-14　指令的执行过程

图 3-14 中的“是否需读取操作数”的分支，表示不是每一条指令都需要到内存中去读取操作数。当然，这不表示指令没有操作的对象，而是操作的对象可能是处理器本身。

以下暂且只讨论仅包括取指令、分析指令（又称指令译码）和执行指令这三个基本步骤时指令的执行方式。

在现代微处理器中，取指令、分析指令和执行指令的工作是由三个部件分别完成的。这三个部件可以同时工作（并行工作），也可以顺序方式工作（串行工作）。

1．顺序工作方式

顺序工作方式是指取指令、指令译码和执行三个部件依次工作，前一个部件工作结束后，下一个部件才开始工作。

顺序工作方式工作过程如图 3-15 所示。在早期计算机系统中均采用这样的执行方式。

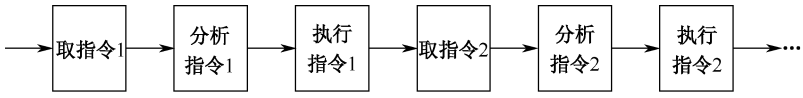


图 3-15　指令顺序执行方式示意图

顺序工作方式的优点是控制系统简单，实现比较容易；另外也节省硬件设备，使成本较低。缺点主要有两个：一是微处理器执行指令的速度比较慢，因为只有在上一条指令执行结束后，才能够执行下一条指令；二是处理器内部各个功能部件的利用率较低。如果以

图 3-14 所示的流程工作，则在取指令部件从内存中读取指令时，分析指令和执行指令部件都处于空闲状态；同样，在指令执行时也不能同时去取指令或分析指令。因此，顺序执行方式时系统总的效率是比较低的，各功能部件不能充分发挥作用。采用顺序方式执行 n 条指令所用时间可用式（3-2）表示为

$$T_0 = \sum_{i=1}^{n}(t_{取指令i} + t_{分析指令i} + t_{执行指令i}) \qquad (3\text{-}2)$$

若假设计算机取指令、分析指令和执行指令所用的时间相等，均为 Δt，则完成一条指令的时间就是 $3\Delta t$，而执行完 n 条指令需要的时间为

$$T_0 = 3n\Delta t \qquad (3\text{-}3)$$

2．并行工作方式

并行工作方式是使上述三个功能部件同时工作，即在指令被取入到处理器、开始进行分析时，取指令部件就可以去取下一条指令；而当指令分析结束开始被执行时，指令分析部件就可以进行下一条指令的译码工作，同时取指令部件又可以再去取新的指令；……。依次进行，在进入稳定状态后，就可以实现多条指令的并行处理。

图 3-16 给出了并行工作方式下的指令执行过程示意图。图中，当第 1 条指令进入指令分析部件时，取指令部件就开始从内存中取第 2 条指令，假如这三个功能部件的执行时间完全相等，均为 Δt，执行第 1 条指令需要的时间为 $3\Delta t$，之后每过一个 Δt 时间，就有一条指令执行完成，则执行 n 条指令所需要的时间为

$$T=3\Delta t+（n-1）\Delta t=（2+n）\Delta t \qquad (3\text{-}4)$$

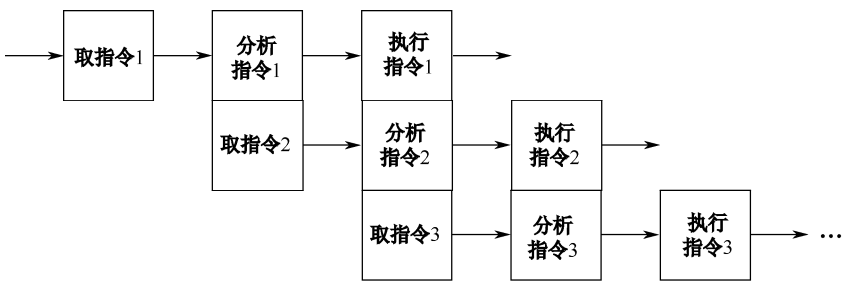


图 3-16　指令并行执行方式示意

由式（3-4）可以看出。与采用顺序执行方式所用的时间 T_0 相比，并行执行方式缩短了系统执行程序的时间，且这种时间上的收益率会随着指令数量 n 的增加而更加显著。

相对于顺序执行方式，并行方式减少的时间量可用系统加速比 S 来描述，即

$$\begin{aligned} S &= T_0 / T \\ &= 3n\Delta t /[(2+n)\Delta t] \\ &= 3n/(2+n) \end{aligned} \qquad (3\text{-}5)$$

例 3-2　某程序段经编译后生成 10 000 条机器指令，假设取指令、分析指令和执行指令所用的时间均为 t。分别求出使用顺序执行方式和并行流水线方式完成该程序段所需的时间，并说明使用顺序执行方式比并行方式慢多少（即系统的加速比）？

由题目知，n=10000，$\Delta t = t$，则由式（3-2），顺序执行完该程序所需时间得

$$T_0=3nt=30\ 000t$$

采用并行流水方式执行该程序需要的时间为

$$T=(2+n)t=10\ 002t$$

顺序执行与并行方式所耗费时间的比为

$$S=T_0/T=30\ 000t/10\ 002t=30\ 000/10\ 002\approx 3$$

可见，顺序执行方式所花费的时间约为并行方式的 3 倍。

图 3-16 所示模型是现代计算机流水线控制技术的基本模型。该模型所给出的是及其理想的情况，即每个部件的工作时间完全相同，也仅在这样的假设下，所示模型的流水线才不会“断流”。这在实际的系统中是不可能的。

为了解决流水线的断流问题，在现代计算机系统中，在取指令和指令译码部分，都设置有指令和数据缓冲栈，可以实现指令和数据的预取和缓存。指令执行部分设置有独立的定点算术逻辑运算部件、浮点运算部件等。另外，加入了预测、分析、多级指令流水线等多项技术，实现对指令和数据的预取和分析，以尽可能地保证流水线的连续。

3.3.2　微型计算机的一般工作过程

计算机的工作过程就是执行程序的过程，也就是逐条执行指令序列的过程。由于每一条指令的执行，都包括取指令（含指令译码）和执行指令两个基本阶段，所以，微机的工作过程，也就是不断地取指令和执行指令的过程。

当需要计算机完成某项任务时，最基本的工作是首先要使用某一种计算机语言[1]编写出相应的程序。编写完成后，需要以文件形式（要起个名字）存放在外存储器中，运行时在操作系统控制下通过接口输入到内存。

进入内存后的程序，会按照逻辑上的顺序依次放入内存各单元。若假设程序已存放到内存，当计算机要从停机状态进入运行状态时，处理器内部的程序计数器（PC）会指向程序的第一条指令。当 PC 所指向的指令被取出后，处理器将自行修改 PC 的值，使其指向下一条指令。指令的执行结果会暂存在内存中，最后在操作系统控制下存入外存或由输出设备送出。图 3-17 给出了程序在进入内存后、计算机按顺序执行方式执行一条指令的工作过程：

- 控制器将要读取指令在内存中的地址赋给 PC（图中假设为 04H），并送到地址寄存器 AR。
- PC 自动加 1，AR 的内容不变。
- 将地址寄存器 AR 的内容发送到地址总线上，并送到内存储器，经地址译码器译码，选中相应的内存单元。
- CPU 的控制器发出“读”控制信号。
- 在读命令控制下，所选中的内存 04H 号单元中的内容（即指令码，图中假设为 97H）被读出送到数据总线上，并送入数据寄存器 DR。
- DR 将读出的指令码送到指令寄存器 IR，然后送指令译码器 ID，进行指令分析。

至此，就完成了一条指令的读取。读取的指令经译码后，若需要再到内存中读取操作数，则继续下述过程：

[1] 关于程序设计语言的相关介绍请参阅本书第 5 章。

- 发送运算所需操作数的地址。
- 读取操作数。
- 使运算器开始执行指令。
- 发送保存运算结果的地址。
- 将运算结果暂存在内存中。

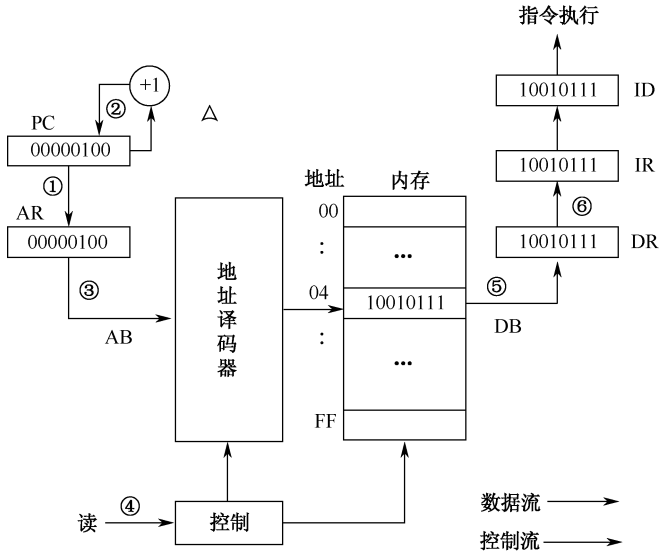


图 3-17　冯・诺依曼计算机工作过程示意图

一条指令执行结束后，就转入下一条指令的取指令阶段。如此周而复始地循环，直到程序中遇到暂停指令方才结束。

上述整个工作过程中，控制器将会发出相应的各种控制信号（如“读”信号、“写”信号等），协调和控制各部件的运行。

应当指出，读操作完成后，04H 单元中的内容 97H 仍保持不变，这种特点称为非破坏性读出（Non Destructive Read Out）。这一特点很重要，因为它允许多次从某个存储单元读出同一内容。

处理器向内存中写入执行结果的过程与“读”操作过程类似，不同的是：此时控制器发出的是“写”命令。CPU 将要写入的内容放到数据总线上；然后发出“写”控制信号，在该信号的控制下，数据被写入指定的存储器单元中。

应当注意，写入操作将破坏该存储单元原存的内容，即由新内容代替了原存内容，原存内容将被清除。

例 3-3　以一个简单的加法运算为例，描述计算机的工作过程。

求解 5+8=？的机器语言程序为

```
机器码
10110000  00000101;第一个操作数（5）送到寄存器
00000100  00001000;5 与第 2 个数（8）相加，结果（13）送到寄存器
11110100;          停机
```

该段程序在内存中的存放形式如图 3-18 所示。由于读取每一条指令都是由一系列相同的操作组成，为简便起见，这里仅给出读取第一条指令的过程描述，如图 3-19 所示。

取第一条指令的过程如下：

- 将指令在内存中的地址（这里为 0000 0000）赋给程序计数器 PC，并送到地址寄存器 AR。
- PC 自动加 1（即由 0000 0000 变为 0000 0001），AR 的内容不变。
- 把地址寄存器 AR 的内容（0000 0000）放在地址总线上，并送至内存储器，经地址译码器译码，选中相应的 0000 0000 单元。
- 控制器发出读命令。
- 在读命令控制下，把所选中的 0000 0000 单元中的内容即第 1 条指令的操作码 1011 0000 读到数据总线。

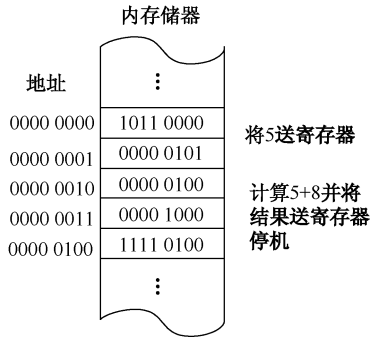


图 3-18　指令在内存中的存放形式

- 把读出的内容 1011 0000 经数据总线送到数据寄存器 DR。
- 取指阶段的最后一步是指令译码。因为取出的是指令的操作码，故数据寄存器 DR 把它送到指令寄存器 IR，然后再送到指令译码器 ID。

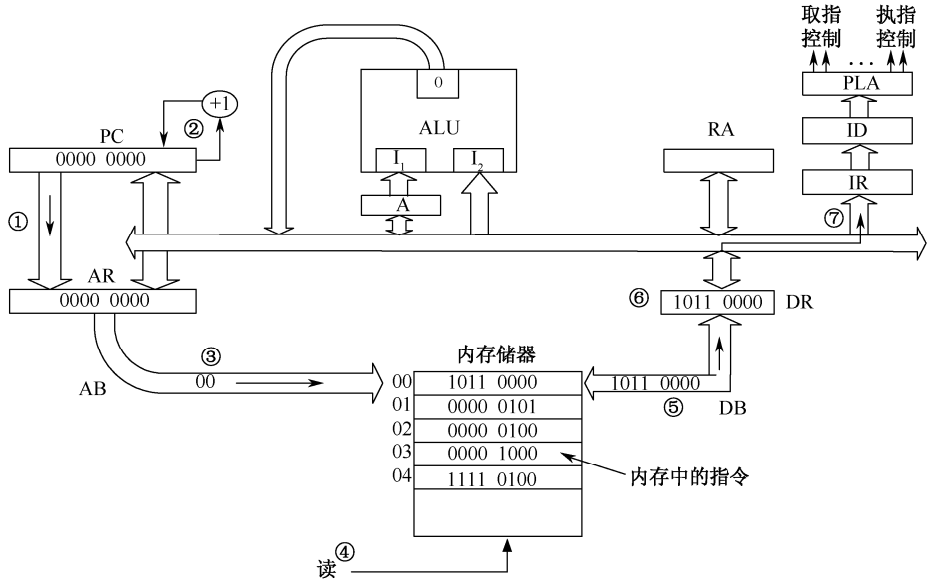


图 3-19　读取第一条指令操作码的过程

读取存放在内存中的操作数的过程与取指令类似，仅第⑦步有不同。上例中，因指令要求读取的操作数要送到寄存器，故由数据寄存器 DR 取出的内容就通过内部数据总线送到寄存器中。由于运算的结果存放在处理器内部的寄存器，所以不需要再访问内存。

但需要注意的一点是，CPU 内部寄存器只能用于数据（中间运算结果）的暂时存放，最终的结果还是需要存放到存储器中。

3.3.3　图灵机与计算机

从以上的叙述，我们对现代计算机的基本工作原理已经有了初步的了解。现在来讨论

一下第 1 章中介绍的图灵机与计算机的关系，希望能够说明（仅仅是说明，而非严格的推理）为什么图灵机模型是现代计算机的理论基础，或者说计算机只是图灵机的翻板。

先来看一个简单示例。

例 3-4 求 3 个正整数 *a*，*b*，*c* 之和。

这是一道小学数学题，求解该题目的方法可以说有无数种。例如：

（1）*a*+*b*+*c*；

（2）*a*+*c*+*b*；

（3）*b*+*c*+*a*；

（4）*a*+*b*+*c*+1−1；

（5）……

很容易看出，不论用上述哪种方法，在 *a*，*b*，*c* 均为正整数的前提下，它们的计算结果都相同，或者说，它们是计算等价的。推而广之，对某一问题，如果用不同的求解方法都能得到相同的正确结果，那么我们就说这些方法是计算等价的。这就如同“条条大路通北京”一样，都从西安出发，走不同的路线，虽然花费的代价（时间、精力等）不同，但最终都到达了北京。这些路线就是计算等价的。如果我们能够说明图灵机与计算机计算等价，那么就可以说计算机只是图灵机的翻板了。

回到例 3-4。已经显然地证明其中的各表达式相对于正整数是计算等价，也就是说既可以用（1）来求解，也可以用（2）求解等。换个角度，就是（1）可以代替（2），（3）也可以代替（1）等。或者说，（1）可以“模仿”（2），（3）也可以“模仿”（1）等。我们将这种“模仿”称为“模拟”。以上各表达式之间就可以相互模拟。所以，计算等价的前提就是要能够相互模拟。即如果集合 A 和 B 能够相互模拟，则 A 和 B 计算等价。以下就从什么是“模拟”入手，从不同角度说明图灵机与计算机的计算等价性。

1．对应的功能部件

计算等价的前提就是要能够相互模拟。那么，什么是“模拟”呢？这是一个很难给出确切定义的名词。可以简单地说，模拟就是一种模仿或是复制。例如，有人冲你做了一个鬼脸，然后你也照着他的样子冲他做了个鬼脸。这是你在模仿他，或者说你对他进行了模拟。考虑一下为什么你能够模拟他。重要的一点：因为他有手，你也有手，你的手对应他的手；他有眼睛，你也有眼睛，你的眼睛对应他的眼睛；你的手、眼睛和嘴的动作对应着他的手、眼睛和嘴的动作……即你们之间存在一系列的对应关系。

因此，A 能够模拟 B 的关键条件是要具有对应关系：如果 A 中元素可以完全对应 B 中的元素，那么 A 就可以模拟 B（注意：这句话隐含的条件是 B 不一定能模拟 A）。反之，如果 B 中元素也可以完全对应 A 中元素，则 B 也就可以模拟 A。

由 1.1.3 节已知，图灵机模型的四要素无限长的纸带、读/写头、输入/输出控制规则及内部状态集合。如果将图灵机设为 A，计算机设为 B，那么，A 中的四大元素在 B 中都具有了一一对应的关系：纸带对应计算机中的存储器（不要想着计算机中的存储器容量是有限的，如果加上光盘等脱机外存，存储容量就可以认为无限大了）；读/写头对应着计算机中的运算器及输入/输出设备；控制规则显然就对应于计算机中的程序；与图灵机一样，计算机中也有状态信息集合。由此可见，计算机与图灵机具备了相互模拟的条件。

2．信息变换（计算）能力

例 3-5　假设有 A 和 B 两个人，A 对 B 做了个鬼脸，但 B 没有冲着 A 做鬼脸，而是将 A 的鬼脸动作记在了日记本上。几天后，C 根据 B 在日记本上的描述记录，冲着其他人做了鬼脸，与 A 的完全一样。

这里，C 将日记本上的文字翻译成了动作，完成了对 A 的模拟。这个“翻译”的过程就是对信息的变换，而变换本身可以理解为一个计算。1.1.1 节中已给出了关于计算的详细描述。广义地讲，计算就是对信息的变换。例如，将 x 变换为 $f(x)$就是一种计算。这里的 x 是输入，$f(x)$则是输出。图灵机是一个计算装置，计算机也是一个计算装置，因为它们都可以将输入信息进行变换后给出相应的输出。

如果将一个图灵机对纸带信息的变换结果输入给另一台图灵机，然后再输入到别的图灵机……，这样相当于对计算进行了组合，从而构成更加复杂的计算。如果一个简单计算对应一个图灵机的话，那么多个简单图灵机就可以构造出复杂的图灵机。最简单的计算就是对 0 和 1 的逻辑运算，任何图灵机都可以把输入和输出信息进行二进制编码，任何一种变换也可以最终分解为对 0 和 1 编码的变换，而对 0、1 编码的所有计算都可以分解为与、或、非三种逻辑运算，即用逻辑电路可以组合出任意的图灵机。

在式（1-1）的图灵机形式化描述中，有一个停机状态 F，它是内部状态 Q 的子集。当图灵机的工作碰到停机状态时就结束计算。

计算机也由各种逻辑电路组合而成，可以进行我们都知道的各种复杂的信息变换，而且在满足一定条件时可以停止。

3．通用性

如果将例 3-5 中的 A 和 B 设为两台图灵机，要使它们能够相互模拟，需要哪些条件呢？

首先，要有一一对应的关系。既然都是图灵机，A 具有图灵机的 4 个要素，B 也同样有；这是宏观的对应。是不是需要 A 和 B 中的状态集合与控制规则都要一样呢？答案是不一定。因为 A 和 B 是否能相互模拟，取决于它们是否计算等价，即对给定的输入，是否具有相同的变换（计算）结果。如果是，则就认为 A 和 B 可以相互模拟。

若设图灵机 A 的输出为 O，假如 B 的输出为 O'，为了使 B 能模拟 A，我们再通过一台图灵机 C，能够将 O' 变换为 O，那么就相当于 B 模拟 A 了，如图 3-20 所示。

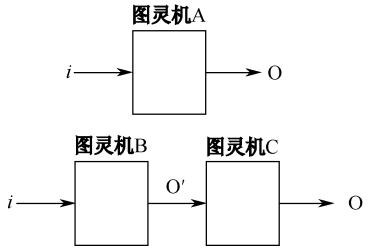


图 3-20　图灵机间的模拟

如果图灵机 A 能够模拟图灵机 B，并且 B 也能模拟 A，则说 A 和 B 是计算等价的。能够模拟其他所有图灵机的图灵机就称为通用图灵机（Universal Turing Machine，UTM），它能接收一段描述其他图灵机的程序，并运行程序实现该程序所描述的算法。这句话的简单解释：任意一台图灵机 TM，其当前的“内部状态、输入数据、输出动作、下一时刻的内部状态”等信息都可以用 0 和 1 的组合（即编码）来表示，这样的一组编码就可以代表 TM。若该 TM 能够将输入 x 变换为输出 y，那么，将 TM 的编码及 x 输入到 UTM 中，则也会得到同样的 y。

以上从功能部件的对应性、信息变换能力及通用性等三个方面叙述了图灵机与计算机

的计算等价性，事实上，现代电子计算机就是这样一种通用图灵机的模拟。图 3-21 给出了一个多（纸）带图灵机模拟计算机的示意图（多带图灵机可以采用固定的模式转换为单带的图灵机，具体的转换方法有兴趣的读者可查阅其他详细介绍图灵机的参考书）。

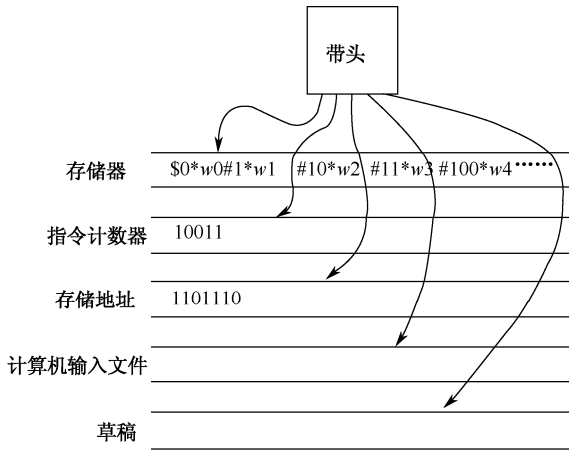


图 3-21　图灵机模拟计算机的示意图

第一条带表示计算机的存储器。假设存储单元的地址按照数值顺序与这些存储单元中的内容交替出现。地址和内容都用二进制书写。符号＊和＃用来表示地址和内容结尾，以及区分二进制串是地址还是内容。另一个标记$表示地址和内容序列的开头，如 0＊表示这里的 0 是地址，而 w0#表示 w0 是存放的内容。

第二条带是“指令计数器”（相当于程序计数器 PC）。这条带保存一个二进制整数，它表示第一条带上的一个存储单元的地址，而该存储单元中的内容是将要执行的下一条计算机指令。

第三条带保存数据的“存储地址”或这个地址的内容（当在第一条带上确定地址位置之后）。为了执行指令，图灵机必须找到一个或多个保存着计算所涉及数据的存储地址的内容。首先，把所需地址复制到第三条带上并与第一条带上的地址比较，直到发现匹配为止。将第一条带上地址中对应的内容复制到第三条带上，并移动到所需要的任何地方，典型情况是，移动到表示计算机寄存器的一个低编号地址。TM 将模拟计算机的指令周期如下：

- 搜索第一条带，寻找与第二条带上指令号匹配的地址。由于第二条带上保存的是下一条要执行的指令的地址，从第一条带上$处开始，向右移动，比较每个地址与第二条带的内容。比较的过程是使带头一前一后地向右移动，并验证扫描的符号是否相同。
- 找到指令地址时检查地址中的内容（指令译码）。由于内容是指令，则其前几个位（指令码）表示要做的动作（如复制、加、分支等），剩余位表示动作中涉及的一个或多个地址（操作数）。
- 如果指令的操作数是地址码（即运算对象的存放地址），则这个地址会被复制到第三条带上。同时将指令的位置标记到第一条带的第二道（在图 3-21 中没有显示出来）上，以便必要时能回到这条指令。
- 执行指令。
- 在执行指令并确定指令不是跳转之后，给第二条带上的指令计数器加 1，再次开始

指令循环。

- 图灵机如何模拟典型计算机，还有许多其他细节。在图 3-21 中显示了第四条带，这条带保存被模拟的计算机输入，因为计算机必须从文件读输入，图灵机可以改为从这条带来读取输入。图 3-21 还显示了一条草稿带。模拟有些计算机指令可能有效地使用一条或多条草稿带来计算诸如乘法的算术运算。

以下是图灵机实现数据传送操作的一个示例描述，读者可以对比一下 3.3.2 中所述微型机的工作过程。

把一个数（源操作数）传送到某个地址（目标地址）中的操作过程如下：

- 从指令中得到这个源操作数的地址。
- 把这个地址写在第三条带上。
- 在第一条带上搜索这个地址，即可找到要传送的数（源操作数）。
- 用同样的方法找到目标地址后，把这个源操作数复制到为目标地址保留的空间里，就实现了“传送”。

如果需要更多的空间来保存这个源操作数，或者源操作数比目标地址中原来的值占用更少的空间，则可以通过平移来改变可用的空间，即

- 把新的值所占之处右边的整个非空白带复制到草稿带上。
- 把新的值写下来，使用这个值的正确的空间数量。
- 把草稿带重新复制到第一条带上，紧接着新值的右边。

还可能出现的一种特殊情形：这个目标地址可能还没有出现在第一条带上，因为在此之前计算机可能还没有用到过它。此时，就在第一条带上找到源数据所属的地方，平移腾出适当的地方，把地址和新的值都保存在这个地方。

最后，假设计算机能够“接收”输出确认指令（可能对应着计算机调用的往输出文件上写 yes 的函数）。当图灵机模拟这条计算机指令的执行时，图灵机进入自身的接收状态并停机。

上面的讨论远远不是完整的形式化的 TM 与计算机相互模拟的证明，但它应当提供了足够的细节来说明图灵机是计算机能够做什么的有效表示。可以将只使用图灵机作为任意种类的计算机的模拟计算装置，通过该装置，更好的研究计算机能计算什么并给出严格的表示。

最后，我们给出断言：计算机能够模拟图灵机，图灵机也能够模拟计算机，即计算机与图灵机是计算等价的。或者说，计算机只是图灵机的翻板。

3.4　非冯·诺依曼计算机

3.4.1　冯·诺依曼计算机的局限性

冯·诺依曼的“存储程序计算机结构”为计算机技术的发展做出了巨大的贡献，几十年来，虽然计算机技术有了迅猛的发展，但传统计算机依然采用的是冯·诺依曼的体系结构。

传统的冯·诺依曼计算机结构属于控制驱动方式。它由存放在内存中的程序指明计算机的操作内容，指令的执行顺序受程序计数器的控制（如 3.3.3 节所述）；也就是说，由指

令控制器控制指令执行的顺序和时机，当它指向某条指令时才驱动该条指令的执行。这种结构的特点是“程序存储，共享数据，顺序执行”。计算中有一条单一的控制流从一条指令传到下一条指令（由程序计数器 PC 提供，执行 *K*、*K*+1、…、*N* 指令），执行指令所需要的操作数通过指令中给定的地址来访问，指令执行结果也通过地址存入一个共享的存储器中。因此，存储程序工作方式的冯·诺依曼计算机本质上是顺序处理机，它的软件和硬件完全分离，适合于对确定的算法和数据进行数值计算，对非数值的处理就显得不足。

随着计算机应用领域的不断拓展，对计算机的性能，特别是对非数值数据的处理提出了更高的要求，如各类三维模型的计算和处理、气象信息处理等，都要求计算机的计算能力能达到万亿次/秒以上。这就使得传统的冯·诺依曼计算机难以满足这种需求，逐渐暴露出了其体系结构上存在的不足，主要表现如下：

- 由于 CPU 与存储器之间会有大量的数据交互，而总线的传输能力却有限。因此，使系统的性能受到了总线传输能力的制约，造成总线瓶颈。
- 按照存储程序原理，指令的执行顺序由程序决定。这就要求在编写程序时必须仔细地分析任务的处理顺序。这对一些大型的、复杂的任务是比较困难的（即需要准确地做好需求分析和模块设计）。
- 由于指令的执行顺序由程序计数器控制，使得即使有关数据已经准备好，也必须逐条执行指令序列。而提高计算机性能的根本方向之一是并行处理，而冯·诺依曼计算机难以实现真正的并行处理。
- 以运算器为中心，I/O 设备与存储器间的数据传送都要经过运算器，使处理效率、特别是对非数值数据的处理效率比较低。
- 冯·诺依曼计算机具有简单的逻辑运算和判断功能，但远不能适应复杂的问题求解和推理的要求。

由于在体系结构上存在以上这些局限，从根本上限制了计算机的发展，特别是并行计算的发展。因此，从 20 世纪 80 年代起，陆续提出了多种与冯·诺依曼计算机截然不同的新概念模型的系统结构，如并行计算机、数据流计算机、量子计算机、生物计算机等非冯·诺依曼计算机，它们部分或完全不同于传统的冯·诺依曼计算机，在很大程度上提高了计算机的计算性能。

*3.4.2 数据流计算机结构

近年来，对非冯·诺依曼计算机的研究主要表现在以下几个方面：

- 在冯·诺依曼体制范畴内，对传统冯·诺依曼计算机进行改造，如采用多个处理部件形成流水处理，依靠时间上的重叠提高处理效率。
- 由多个运算部件组成阵列机结构，形成单指令流多数据流，提高处理速度。
- 用多个冯·诺依曼计算机组成多机系统，支持并行算法结构。
- 以上这些研究已比较成熟。而第四种结构可称为真正的非冯·诺依曼计算机结构。它是从根本上改变冯·诺依曼机的控制流驱动方式。例如，采用数据流驱动工作方式的数据流计算机，只要数据已经准备好，有关的指令就可并行地执行。它为并行处理开辟了新的前景。

图 3-22 所示是日本 NEC 公司在 1984 年推出的一种非冯·诺依曼计算机结构 DIPS

（Dataflow Image Processing System），图 3-22 中的高速数据流处理机采用数据驱动方式（不是冯・诺依曼计算机的控制驱动），即指令的执行是由数据驱动的。当指令具有了所需要的操作数据且输出端没有数据时，指令就可以被执行。一旦指令开始执行，其输入端的数据就取消，执行结果会被送到输出端。

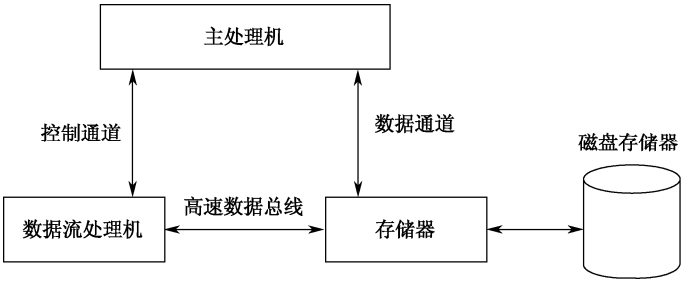


图 3-22　非冯・诺依曼计算机结构示意图

这种结构使得程序的执行顺序不是由程序计数器控制，而是由指令间的数据流控制。程序变成了由指令连接的有向图。这样，就实现了被处理的数据一到，就会立刻被处理，从而解决了冯・诺依曼计算机中的总线瓶颈问题。

图 3-23 所示为数据流处理机工作原理示意图。图 3-23 中，数据流处理机由运算模块和访问模块组成，两者间通过一个接口相连。各指令的执行部件和存储器的读/写操作部件通过数据总线连接成环形，以流水方式工作。

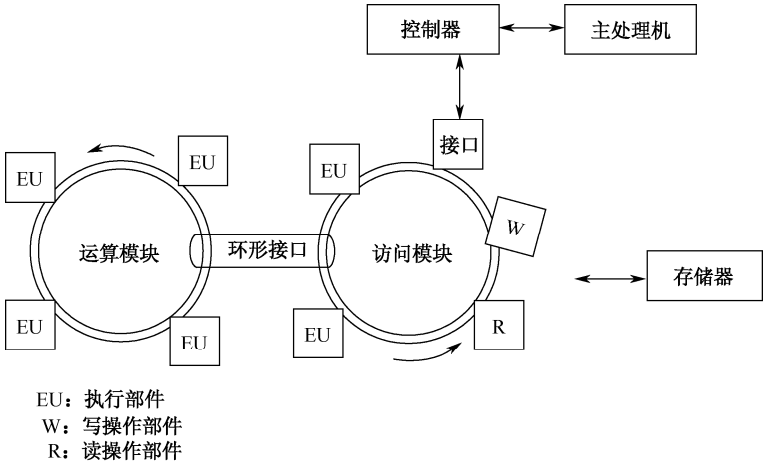


图 3-23　数据流处理机工作原理示意图

当访问部件从存储器中读数据时，读取部件中的程序会对读出的数据赋予数据名并加上要到达的目的地址，然后将其放到数据总线上流动。环形总线上的每个指令执行部件不断检测总线上的数据流，当发现是发向本部件的数据时就立刻捕获并进行处理。处理完成后再为其加上新的数据名和目标地址。

由于数据是在执行部件间流动的过程中被处理，在此期间处理器和存储器之间不再需要进行输入/输出操作，所以，数据流处理机的处理能力受存储器传输速度的影响较小。另外，由于数据是由不同的执行部件来标记应送往哪个执行部件执行，以及执行结果的去向（即目标地址），而不是按存储地址标识，从而摆脱了传统计算机按照程序规定的顺序执行

操作的束缚，实现了数据的并行处理。

非冯·诺依曼计算机结构的主要特征是并行性，而实现的主要手段是以存储器为核心数据驱动。同时，相关的研究还集中在具有逻辑推理和问题求解功能、能够识别自然语言及各种多媒体信息的智能接口等方面。

3.4.3 哈佛结构

冯·诺依曼计算机采用“存储程序”工作原理、以运算器为核心，具有一个存储器、一个控制器、一个运算器及输入和输出设备。其典型的结构模型如图 3-13 所示，即所有的输入和输出都需要通过运算器。人们将这种结构称为冯·诺依曼结构，又称普林斯顿（Princeton Architecture）结构。

由于计算机采用二进制，指令和数据都用二进制码表示，指令和操作数的地址又紧密相关。因此，很自然地采用了将指令和数据统一存放、共享同一总线的结构。但这种结构使得信息流的传输只能采用串行方式，从而影响了计算机性能的提高。例如，我们已经知道，完成一条指令的执行需要经过取指令、指令译码、读取操作数、执行和存放结果这样五个步骤。由于指令和数据存放在同一存储器中、共用同一条总线（数据总线）传输，使取指令时就必然无法同时读取操作数。因此，它们无法重叠执行，只能串行执行。

哈佛结构（Harvard Architecture）是一种并行体系结构，其结构模型如图 3-24 所示。它将程序指令和数据分开存储在不同的存储空间中，即程序存储器和数据存储器是两个独立的存储器，每个存储器独立编址、独立访问。相应地就有 4 条系统总线：用于传送指令的数据总线和地址总线，以及用于传送数据的数据总线和地址总线。这种分离的程序总线和数据总线可以允许在一个机器周期内同时获得指令操作码（来自程序存储器）和操作数（来自数据存储器），使数据的吞吐率提高了 1 倍，从而提高了计算机的执行速度。另外，由于程序和数据存储在两个分开的物理空间中，因此，取指令和存取操作数可以重叠执行。处理器首先到指令存储器中读取指令，经过译码后得到数据地址，再到相应的数据存储器中读取数据，并进行下一步执行指令的操作。

总之，与冯·诺依曼结构相比，哈佛结构具有如下两个显著特点：

- 使用两个独立的存储器模块，分别存放指令和数据。每个模块中都不允许指令和数据并存。
- 使用独立的两组总线，分别作为 CPU 与存储器之间的专用指令和数据的通信通道。这两组总线间毫无关联。

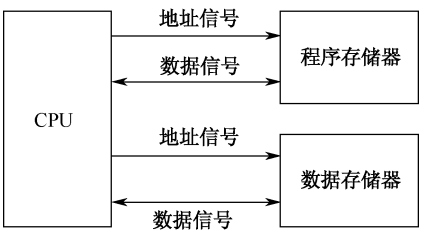


图 3-24　哈佛结构模型

在改进的哈佛结构中，将图 3-24 中的两组总线合并为一组，公共地址总线用于访问两个存储器模块，公共数据总线则被用来完成程序存储器或数据存储器与 CPU 之间的数据传输；两条总线由程序存储器和数据存储器分时共用。

在现代处理器中，程序存储器和数据存储器均采用 Cache，省去了从主存储器中读取指令和数据的时间，大大提高了运行速度。

冯·诺依曼结构和哈佛结构的主要区别就在于程序空间和数据空间是否为一体。前者

将两个空间重合，而后者是分开的。

冯·诺依曼计算机简单、低成本的总线结构，造就了计算机，特别是微型计算机的迅速发展。目前，虽然已有众多关于非冯·诺依曼计算机的研究，但数据流计算机主要用于大型机系统中，其他如量子计算机、生物计算机等都还处于实验室研究阶段，离进入市场还有相当的距离。

在现代微型计算机系统中，其总体结构依然是冯·诺依曼结构，但在微处理器内部，由于采用 Cache 技术，实现了指令和数据分开存放，同时共享公共总线，属于改进型的哈佛结构。与冯·诺依曼结构相比，哈佛结构复杂度比较高，对外围设备的连接和处理要求较高，不适合于存储器扩展。所以，除了在 CPU 内部之外，哈佛结构主要应用于单片机和微控制器中，如 Intel 公司的 51 系列、Microship 公司的 PIC16、ARM 公司的 ARM9～ARM11 等。

3.5　操作系统

操作系统（Operating System，OS）是管理计算机硬件与软件资源的程序，同时也是计算机系统的内核与基石。没有它，今天的计算机可以说是完全没有意义的。

3.5.1　操作系统概述

操作系统在计算机系统中的作用可以相当于“大脑”在人体中的作用。不论这种比喻是否恰当，但至少说明了一个问题——操作系统对计算机系统而言是至关重要的。

1．什么是操作系统

首先，操作系统是一个程序，虽然它非常庞大和复杂，但它依然只是一个程序，是控制其他程序运行、管理系统资源并为用户提供操作界面的系统软件。

操作系统由一系列具有不同管理和控制功能的程序模块组成，位于硬件和用户之间，是覆盖于计算机硬件系统上的第一层软件。它一方面为用户提供接口，方便用户使用计算机；另一方面它能管理计算机软硬件资源，以便合理充分地利用它们。

操作系统的作用可以用图 3-25 示意，总体上如下：

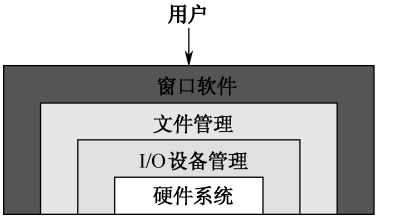


图 3-25　操作系统的作用示意图

- 隐藏硬件。由于直接对计算机硬件进行操作非常困难和复杂，因此，从用户的角度，需要计算机具有友好、易操作的使用平台。
- 为用户和计算机之间的“交流”提供统一的界面，使用户不必考虑不同硬件系统可能存在的差异。
- 管理系统资源。计算机系统中的主要资源有处理器、存储器、I/O 设备、运行的数据和程序。资源管理主要就是对以上四种资源有效地进行的管理和分配，使有限的系统资源能够发挥更大的作用。

早期的计算机中没有操作系统，用户在计算机上的操作，完全由手工进行，采用绝对的机器语言（二进制代码）形式编写程序，通过接插板或开关板控制计算机操作。这个时期的计算机只能一个个、一道道地串行算题，一个用户上机，就独占了全机资源，使资源

利用率和效率都很低。

晶体管的诞生使得计算机产生了一次革命性的变革。操作系统的初级阶段是监控程序和批处理程序。在 20 世纪 60 年代早期，商用电脑制造商制造了批处理系统，但此时，不同型号的计算机具有不同的操作系统，无通用性。

1964 年，第一代共享型而非每种产品量身订做的操作系统 OS/360 诞生，它可以运行在当时 IBM 的系列大型计算机上。

随着计算机技术的发展，操作系统的功能越来越强大。今天的操作系统已有包括分时、实时、并行、网络及嵌入式操作系统等多种类型，成为不论是大型机、小型机，还是微型机都必须安装的系统软件。

2．操作系统的分类

对 OS 进行严格的分类是困难的。早期的 OS，按用户使用的操作环境和功能特征的不同，可分为三种基本类型：批处理系统、分时系统和实时系统。随着计算机体系结构的发展，又出现了嵌入式 OS、分布式 OS、个人计算机 OS 和网络 OS。

目前的操作系统种类繁多，很难用单一标准统一分类。若从应用领域划分，可分为桌面操作系统、服务器操作系统、主机操作系统和嵌入式操作系统等；若根据所支持的用户数目，可分为单用户系统（如 Windows、MS-DOS 等）和多用户系统（如 UNIX 等）；从硬件结构的角度，可分为网络操作系统（如 Netware、Windows NT 等)、分布式操作系统（如 Amoeba 等）和多媒体操作系统（如 Amiga 等）。除此之外，还可以从源码开放程度、使用环境、技术复杂程度等多种不同角度进行分类。下边简要介绍一下几种类型的操作系统。

1）分时操作系统

分时操作系统（Time-sharing Operating System）是指多用户通过终端共享一台主机 CPU 的工作方式。为使一个 CPU 为多道程序服务，将 CPU 划分为很小的时间片，采用循环轮作方式将这些 CPU 时间片分配给排队队列中等待处理的每个程序，如图 3-26 所示。由于时间片划分得很短，循环执行得很快，使得每个程序都能得到了 CPU 的响应，好像在独享 CPU。分时 OS 的主要特点是允许多个用户同时运行多个程序；每个程序都是独立操作、独立运行、互不干涉。现代通用 OS 中都采用了分时处理技术。例如，UNIX 是一个典型的分时 OS。

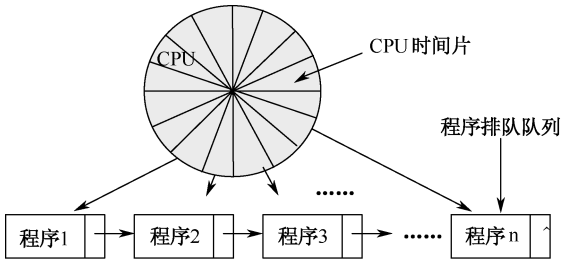


图 3-26　分 6 时占用 CPU 时间片示意图

2）网络操作系统

网络操作系统（Net Operating System，NOS）是向网络计算机提供服务的特殊的操作系统。它在计算机操作系统下工作，使计算机操作系统增加了网络操作所需要的能力。NOS

运行在网络中称为服务器的计算机上，并由联网的计算机用户（客户端）共享，它的功能包括网络管理、通信、安全、资源共享和各种网络应用。网络 OS 的目标是用户可以突破地理条件的限制，方便地使用远程计算机资源，实现网络环境下计算机之间的通信和资源共享。例如，Novell Netware 和 Windows NT 就是网络 OS。

3）分布式操作系统

分布式操作系统（Distributed Software Systems）是指通过网络将大量计算机连接在一起，以获取极高的运算能力、广泛的数据共享以及实现分散资源管理等功能为目的的一种 OS。它的优点：①分布性。它集各分散结点计算机资源为一体，以较低的成本获取较高的运算性能。②可靠性。由于在整个系统中有多个 CPU 系统，因此，当某一个 CPU 系统发生故障时，整个系统仍旧能够工作。显然，在对可靠性有特殊要求的应用场合可选用分布式 OS。

4）个人计算机操作系统

个人计算机 OS 是一种单用户的 OS。它的特点是计算机在某一时间为单个用户服务；现代个人计算机 OS 采用图形界面人—机交互方式操作，用户界面友好，用户无须学习专业理论知识，就可以掌握对计算机的操纵。典型的个人计算机 OS 是 Windows。

3．操作系统的功能

操作系统的职能是负责系统中软硬件资源的管理，合理地组织计算机的工作流程，并为用户提供一个良好的工作环境和友好的使用界面。

从资源管理角度看，操作系统具有五大基本功能，如图 3-27 所示：

- **进程管理**。又称为作业管理或处理器管理，其主要任务是对处理器的时间进行合理分配、对处理器的运行实施有效的管理。
- **存储器管理**。由于多道程序共享内存资源，所以，存储器管理主要任务是对存储器进行分配、保护和扩充。
- **文件管理**。有效地管理文件的存储空间，合理地组织和管理文件系统，为文件访问和文件保护提供更有效的方法及手段。
- **设备管理**。根据确定的设备分配原则对设备进行分配，使设备与主机能够并行工作，为用户提供良好的设备使用界面。
- **用户接口**。用户操作计算机的界面称为用户接口（或用户界面），用户通过命令接口或程序接口 API（Application Programming Interface），实现各种复杂的应用处理。

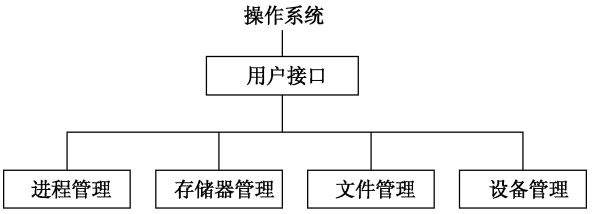


图 3-27　操作系统功能示意图

3.5.2　进程管理

进程管理又称处理机管理或作业管理，其主要任务是对处理机进行分配（解决谁来使

用处理机和怎样使用处理机的问题），并对运行进行有效的控制和管理。在操作系统中，把用户请求处理机完成一项完整的工作任务称为一个作业。当有多个用户同时要求使用处理机时，允许哪些作业进入、不允许哪些进入、怎样安排已进入作业的执行顺序等，这些就是处理机管理模块的任务。由于在多道程序环境下[1]，处理机的分配和运行都是以进程为基本单位的，所以，对处理机的管理最终可以归结为对进程的管理，包括进程控制、进程同步、进程通信和进程调度。作为入门级教材，本节将仅涉及进程管理的一般描述，对详细的控制过程和算法请参阅“操作系统”相关书籍。

要理解进程管理，首先要了解“进程”这个概念，而要弄清楚什么是进程，我们又需要从程序的执行方式讲起。

1．程序的执行方式

由 3.3.1 节已知，程序是实现某一特定功能的指令序列。这里的“功能”可能是由若干个“小功能（程序段）”组成的。程序在执行时，必须按照一定的次序，只有在前一个程序段执行完，后一个程序段才能进行。例如，只有输入了用户程序和数据，才能进行计算（处理），然后才能输出。没有第 1 步，也就无法进行第 2 步。以下为了叙述方便，我们假设一个程序中包括输入（I）、计算（C）和打印输出（P）等三个程序段，用结点（Node）表示程序段的操作，则程序的执行过程如图 3-28 所示。

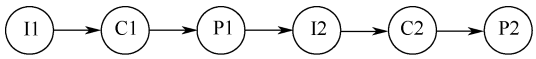


图 3-28　程序顺序执行方式示意图

图 3-28 中的输入、计算和打印输出三个程序段之间，存在着严格的执行顺序，只有输入了信息才能进行计算，只有计算产生了结果才能输出，即对一个作业，存在着 I1→C1→P1 这样的前趋关系，必须按顺序执行。

例 3-6　假设有 3 条语句的程序如下：

```
S1：a=x+y
S2：b=a+4
S3：c=b+1
```

该 3 条语句必须按 S1→S2→S3 顺序执行。因为只有在 a 被赋值后才能执行 S2，b 被赋值后才能执行 S3。

图 3-28 所示的方式称为程序的顺序执行方式。顺序执行时，任一时刻系统中只有一个程序在执行，程序工作于封闭环境中，独占系统的全部资源，只要环境和初始条件相同，程序无论执行多少遍，结果都是一样的。所以，顺序执行具有顺序性、封闭性、结果可再现性等特点。这种特性很便于程序员对程序的检测和校正，但降低了资源的利用率和系统的处理效率。

虽然顺序执行有它独有的优点，但确难以适应现代多任务系统的要求。再来看图 3-28。对一个作业，执行过程存在 I1→C1→P1 这样的顺序，但并不说明存在 Pi→Ii+1 的关系，也就说并不是第 2 个作业的输入必须取决于第 1 个作业的输出（这个前提很重要）。因此，在对多个作业进行处理时，我们可以不按照图 3-28 所示的顺序执行方式，而是在第 1 个程序

[1] 多道程序环境是指在计算机中有多个程序在同时运行，又称并发执行。现代计算机都采用多道程序并发执行方式。

段输入后开始进入计算阶段时，就可以输入第 2 个程序段，第 1 个程序段计算完开始输出时，就可以同时开始计算第 2 个程序段，并输入第 3 个程序段，……。这样多道程序同时执行的方式称为并发执行方式，可用图 3-29 表示。

图 3-29 说明，Ii+1 和 Ci 及 Pi−1 是重叠的，即它们是可以同时执行的。对这种并发执行方式我们也来看一个示例。

例 3-7　执行以下程序语句：

```
S1：a=x+2
S2：b=y+4
S3：c=z+9
S4：d=a+b+c
```

可以看出，S1、S2 和 S3 可以同时（并发）执行，因为它们彼此互不依赖。只有 S4 必须在 S1、S2 和 S3 执行结束后才能之后（必须要等 a、b、c 被赋值）。该程序段的执行也可以用图 3-30 表示。

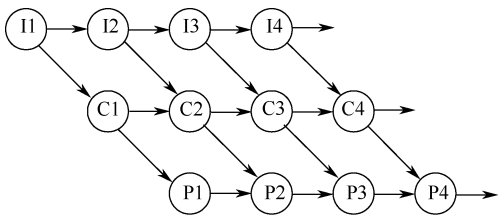


图 3-29　程序并发执行方式示意图

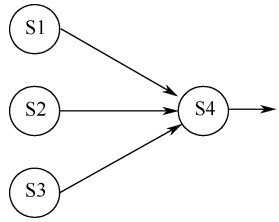


图 3-30　例 3-7 程序执行示意图

程序的并发执行，提高了系统的效率，但也产生了一些与顺序执行不同的新特征。首先，由于多个程序并发执行，整个系统资源为多程序共享，这样虽然提高了资源的利用率，但却存在多个程序对资源的竞争和相互制约问题。例如，打印机被 P1 占用，P2 就不能再使用，只好等待，因此，就存在“走走停停”的情况。由于多道程序共享系统中的各类资源，因而这些资源的状态将由多个程序来改变，程序的运行环境不再具有“封闭”性，也即程序的执行会受到其他程序的影响，从而也导致失去了可再现性，造成多个程序可能因被执行的先后顺序不同而得到不同的结果。

例 3-8　有两个循环执行的程序 A 和 B，共享一个变量 N。程序 A 每执行一次，都要做“N=N+1”的操作，程序 B 每执行一次都要做：“输出 N，N=0”操作。设某时刻 N=n，且 A 和 B 以不同的速度运行，则可能会出现以下三种情况：

（1）先运行 A，再运行 B，得 N 值为

```
n+1，n+1，0
```

（2）先运行 B，再运行 A，得 N 值为

```
n，0，1
```

（3）先运行 B 的“输出 N”，再运行 A，之后运行 B 的 N=0，得 N 值为

```
n，n+1，0
```

例 3-8 的执行结果说明，程序在并发执行时，由于失去了封闭性，其计算结果就不再如顺序执行方式那样可再现，而变得与它的执行速度有关，即程序在经过多次执行后，虽然每次执行的环境和初始条件都一样，但却会得到不同的结果。

2．进程（Process）

为了使程序在多道程序环境下能并发执行，并确保“可再现性”，1966 年，美国麻省理工学院的 J • H • Sallexer 提出了“进程”的概念。引入这一概念的目的就是为了解决现代计算机中多道程序共享系统资源的问题。

“进程”被定义为“可并发执行的程序在一个数据集合上的运行过程”（这句话看上去有点高深）。简单地说，进程就是执行起来的程序。程序本身是静态的（编写好了可以存放在磁盘上不动），但进程是动态的，是“活”着的程序。有“活”当然就有“亡”，所以，进程是有生命周期的。进程的主要特征如下：

- **动态性**。这是进程最基本的特征。表现在因创建而产生，由调度而执行，因得不到所需资源而暂停，因被撤销而消亡。
- **并发性**。这是进程的重要特征，也是操作系统的重要特征。并发性是指多个进程同存于内存中，能在同一段时间内同时运行。
- **独立性**。进程是一个能独立运行的基本单位，也是进行资源分配和调度的独立单位。
- **异步性**。每个进程都按独立的、不可预知的速度向前推进，即各进程并不是按相同的速度运行，而是异步运行，这一特征就导致了程序执行的不可再现性（如例 3-8）。为了避免这一点，操作系统中专门设置了一个称为“进程控制块”的数据结构，它负责记录进程的所有相关信息，保证进程从暂停到重新运行时能获得其暂停时的状态。

打开 Windows OS 中的任务管理器，我们就可以看到多个进程的运行情况。在图 3-31 中，左侧为启动的 Word、Excel 和计算器等 3 个应用程序，右侧是系统运行的进程列表，有系统程序进程，也有应用程序进程，其中椭圆线中的是左侧 3 个应用程序对相的进程。

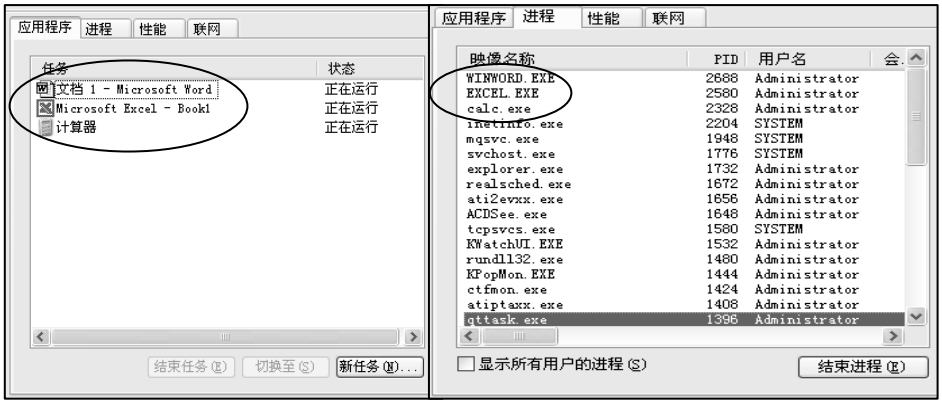


图 3-31　Windows 任务管理器中的应用程序及相应的进程列表

3．进程的基本状态

上文已谈到，进程是“活着的程序”，因此，它具有生存周期。在它“活着”的时间里，并不是始终处于运行状态（因为系统中不是只有一个进程），它会受到资源（如 I/O 请求等）的制约。如果它需要的资源正被其他进程占用，它就只能停下来等待了。所以，进程的执行过程是间断性的，其状态处于不断变化中。通常，一个进程必须具有以下三种基本状态（图 3-32）：

- **就绪（Ready）状态**。进程已经获得了除 CPU 之外所必需的一切资源，一旦分配到 CPU，就可以立即执行（“万事具备，只欠东风”）。在多道程序环境下，可能有多个处于就绪状态的进程，通常将它们排成一队，称为就绪队列。
- **运行状态**。进程获得了 CPU 及其他一切所需资源，正在运行。对单个 CPU 系统而言，只能有一个进程处于运行状态；在多处理机系统中，则可能有多个进程处于运行状态。
- **等待状态**。由于某种资源得不到满足，进程运行受阻，处于暂停状态，等待分配到所需资源后，再投入运行。处于等待状态的进程也可能有多个，也将它们组成排队队列。

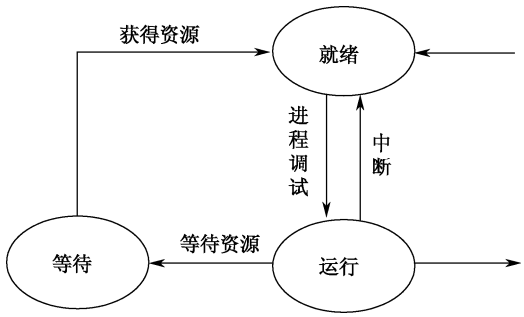


图 3-32　进程的三种基本状态

4．进程控制块

一个进程由三部分组成：程序块、数据集合及进程控制块 PCB（Process Control Block）。程序块就是程序本身，用于描述进程所要完成的操作；数据集合包括进程执行时所需要的数据集和工作区；PCB 是进程控制的核心机制，其中记录了描述进程情况及控制进程运行所需的各种信息，操作系统正是根据 PCB 中的信息对并发执行的进程进行控制和管理。一般来讲，PCB 中通常包含以下信息：

- **进程标识信息**。包括进程的名称、进程标识符等，用于唯一区别于其他进程。
- **进程调度信息**。包括进程状态和进程优先级。进程状态用来记录进程当前的状态，作为进程调度和分配系统资源的依据；进程优先级用于在进程调度过程中，哪一个进程可以优先获得 CPU 资源。
- **进程现场信息**。记录进程因放弃 CPU，而必须保存的现场信息，作为再次恢复运行时从断点处开始执行的依据，如寄存器状态、程序计数器值和中间结果值等（有了这样的“现场信息”，再加上适当的进程同步控制手段，就不会出现例 3-8 那样的“不可再现”的情况了）。
- **进程控制信息**。包括如构成进程的程序和数据在存储器中的地址（以便再次调度执行进程时能够找到其程序和数据）、该进程所需要的全部资源和已经分配得到的所有资源清单（不包括 CPU）、本进程所在的排队队列中下一个进程的 PCB 入口地址、进程同步和通信机制信息等。

每当系统创建一个新进程时，都会自动为其建立一个 PCB，然后为进程分配相应的资源，并将其插入就绪队列。

引入进程控制块的主要目的就是为了随时获得进程的各种信息，为进程控制和调度提

供依据，以避免多道程序并发执行时出现的不可再现问题。

5．进程控制和调度

进程控制的主要任务是调度和管理进程从“创建”到“消亡”整个生存周期过程中的所有活动，包括创建进程、转变进程的状态、执行进程、撤销进程等操作。

我们已经知道，进程具有就绪、运行和等待三种基本状态，在多数操作系统中还增加了“新状态”和“终止状态”两种。进程在它的整个生命周期中，就是不断地从一个状态转换到另一个状态，直到运行完成或出现异常结束，才进入终止状态。进程状态的转换过程可以简述如下：

- OS 创建一个新进程，先为其分配相应的资源，并使其处于“新状态”。
- 当就绪队列能够接纳新进程时（如有空位了），OS 就将其送入就绪队列，处于就绪状态。
- OS 根据进程在就绪队列中的位置、优先级等（这些都在 PCB 中）信息及进程调度算法，为进程分配 CPU，从而使其转入运行状态（此时的进程称为当前进程）。
- 运行中的进程可能因所需资源不能得到（或许该资源正在被其他进程使用）而无法执行，则转入等待状态。
- 运行中的进程也有可能虽然拥有所需的全部资源，但分给它的 CPU 时间用完了，或有更高优先级的进程出现，则只好退出处理机，转入就绪队列。
- 执行完成或出现异常，则进程转入终止状态。

一个进程可以在三种基本状态之间多次转换，但新状态及终止状态只能出现一次。基于时间片轮转法的进程状态转换过程如图 3-33 所示。

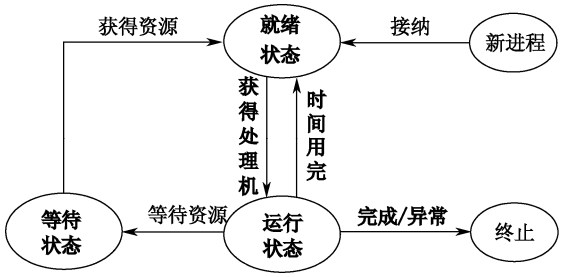


图 3-33　进程状态转移示意图

“进程调度”就是根据具体情况为每个进程分配 CPU。不同的 OS 所采用的调度方式（算法）不完全一样，其中有一种方式就是“分时”原理，或称为时间片轮换法。它所产生的最终效果是使多个正在运行的程序（进程）看上去像是在同时运行一样。因为系统中只有一块 CPU（这里不考虑多处理器系统的情况），因此，这种“同时”只是一种假象，在给定的一个小时间段里，真正能够运行的只有一个进程。

那么，这种“假象”是如何获得的呢？答案：让所有的进程轮换着进入 CPU 运行，每个进程轮流占用处理器一段很小的时间（图 3-26），时间到了就退出等待，直到下次轮到再继续。

采用时间片轮换调度算法的系统中，每个进程都被分配一定的 CPU 使用时间，就绪队列中的进程按先来先服务原则，每次调度时，系统都将 CPU 分给排在队首的进程一段时间。若在此时间内该进程运行完毕，则转入终止状态；若虽未运行结束但 CPU 使用时间到，则

转入就绪状态（排到队尾），直到下次轮到再继续；若虽然 CPU 使用时间还有，但所需资源缺乏，则转入等待状态，直到重新获得资源，再进入就绪队列。

例如，假设有 A、B、C、D 4 个进程，规定每个进程占用 CPU（资源）运行 20ms。进程 A 首先披挂上阵，运行 20ms，不论是否运行结束，都必须让出 CPU 给 B，此时 A 就处于运行挂起（就绪）状态；B 占有 CPU 运行 20ms 后转为挂起，将处理器让给 C；之后是 D。然后再开始 A 的第二轮，……。如此循环往复，直到 4 个进程都运行结束为止。

时间片轮换调度算法可以保证就绪队列中的所有进程在给定时间内，均能获得一个时间片的 CPU。由于相对人的感知能力，每个进程所占用的 CPU 时间都很短，使得整个轮换过程进行得非常之快，我们无法感觉到进程的“运行”、“挂起[1]”、“运行”、……，就好像每个程序都在同时运行一样（虽然它们实际上平均只有 1/4 的时间在运行）。

OS 进行进程调度的依据是 PCB。当要调度某个进程执行时，首先从该进程的 PCB 中查出它的现行状态（是就绪、运行还是等待？）和优先级，并以此作为是否执行该进程的调度依据；在调度到某进程后，要根据其 PCB 中所保存的现场信息，恢复其中断前的寄存器、程序计数器等现场信息，并根据程序和数据的地址，找到要继续执行的断点处；在进程执行过程中，当需要和其他进程实现同步、通信或访问文件时，也要依据相应的进程控制信息；当进程因故而暂停执行时，又要在 PCB 中保存中断现场信息。当某进程结束时，则删除其 PCB。

3.5.3 存储器管理

虽然计算机中存储器的容量随着技术的发展一直在不断地扩大，但仍然不能满足现代软件发展的需求，存储器特别是内存依然是一种宝贵而紧俏的资源。对其的有效管理，不仅影响到存储器的利用率，也对系统的整体性能有重大的影响。

进程管理解决多道程序的并发执行问题，存储器管理解决的是内存的分配、保护和扩充问题。计算机中的作业（处理的数据和程序）首先存放在外存中，在使用（运行）时则需要调入内存（可以考虑一下需要调入内存的原因）。由 3.1.2 节已知，外存和内存在存储容量上相差很大。因此，就引出了这样一些问题：程序是如何调入内存的？将调入到内存的什么地方？如果内存不够用该如何处理？当有多道程序运行时，如何分配内存空间才能最大限度地利用有限的内存为多道程序服务？等等，这些就是 OS 中存储器管理程序要解决的问题。具体如下：

- **存储分配**。为每个作业按一定策略和算法分配所需的内存空间。
- **地址变换**。将程序在外存空间中的逻辑地址转换为在内存空间中的物理地址。
- **存储保护**。保护各类程序（系统的、用户的、应用程序的）及数据区免遭破坏。
- **存储扩充**。解决在小存储空间中运行大程序的问题。

在进一步学习存储管理功能之前，我们先了解一下程序装入内存的过程。

1．程序的装入和链接

在多道程序环境中，程序要运行必须要先为之创建进程，而创建进程的第一件事，就

[1] “挂起”是进程 5 种状态之外的又一种状态。表示暂停的意思，挂起状态又称为静止状态。相应地，非挂起状态就称为活动状态。进程可以由就绪状态转为挂起（静止就绪），也可以由等待状态或执行状态转为挂起。执行状态转为挂起则进入静止就绪状态。

是将程序和相应的数据装入内存。

我们用程序设计语言（无论哪种语言）编写的、完成某项特定的任务的程序称为源程序。源程序编写完成时，都首先存放在外存中，要使其能够被执行，需要装入内存。要将一个用户源程序变为在内存中的可执行程序，通常需要经过以下几步（图 3-34）：

- **编译**（Compile）。由编译程序将源程序转换为若干个由二进制机器语言表示的目标（Object）程序模块。
- **链接**（Link）。将目标程序模块及它们所需要的库函数（已编好有的、具一定功能、可供其他程序引用的程序段）链接在一起，形成装入模块（Load Module）。
- **装入**（Load）。将装入模块装入内存。

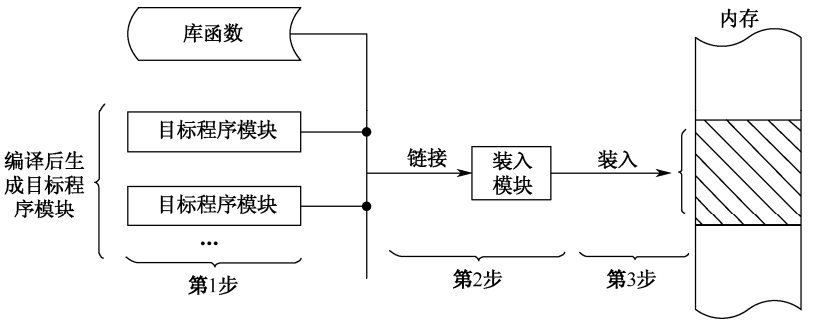


图 3-34　从源程序到装入内存的处理过程

2．存储分配

在程序中，常常需要明确到指令操作的数据或下一条要执行的指令的存放地址（或者说到哪里去找它们）。编写程序时，这些地址都是用字符或字符串表示的，称为符号地址。源程序经编译后，符号地址被转换为用二进制码表示的相对地址（不是实际运行的地址）。当程序运行要装入内存成为进程时，则要将相对地址转换成主存中的物理地址。CPU 在访问内存时，根据物理地址进行数据或指令的读和写。存储地址的这个变换过程可用图 3-35 表示。

系统在将每道用户作业（程序）从外存调入内存时，就为其创建进程、分配相应的存储空间及必要的资源，并送入就绪队列。将程序装入内存的方式有三种：

- **绝对装入**。根据装入模块中的地址，将程序和数据装入到内存中事先指定的位置。这种方法在程序装入前即要确定所需的内存空间和地址，适合单道程序系统，在多道程序系统中难以实现。
- **可重定位装入**。由装入程序根据内存当时的实际情况，将程序装入到内存适当的地方，并将相对地址转换为绝对地址（图 3-35（c））。该方法可用于多道程序系统，但它不允许程序在内存中移动位置。由于一个进程可能会被多次换入或换出内存，每次换入后的位置不一定相同，此时就必须采用动态装入方法。
- **动态装入**。根据内存的实际情况将程序装入到适当的地方，但要到真正运行时才进行相对地址到绝对地址的转换。现代 OS 多采用动态重定位装入法。

程序装入内存时成为进程，当进程运行完毕后，系统需要将其所占用的内存空间收回，以便装入下一道程序。

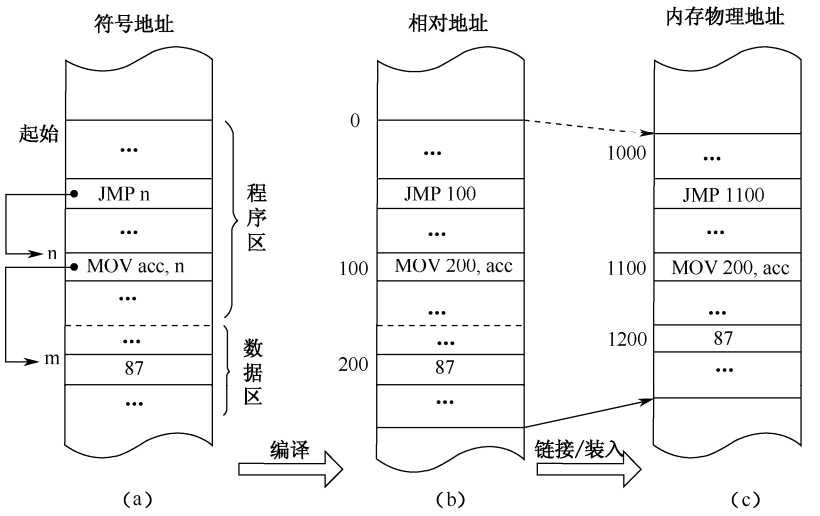


图 3-35　程序装入内存过程示意图

3．存储保护

在计算机中运行的系统程序、应用程序和用户程序都存放在主存中。为了确保各类程序在各自的存储区内独立运行，互不干扰，系统必须提供安全保护功能。措施之一就是把各类程序的实际使用区域分隔开，使得各类程序之间不可能发生有意或无意的损害行为。这种分割是靠硬件实现的。用户程序只能使用用户区域的存储空间，而系统程序则使用系统区域的存储空间。

4．存储扩充

程序只有装入内存才能运行。对一个大程序来讲，若要将其全部装入，则需要较大的内存空间。另外，在多道程序系统中，可能有大量的程序要求运行，但由于内存空间不足以容纳所有希望运行的程序，只能将大量的程序留在外存中。程序从外存装入内存需要一定的时间（这个时间相对于 CPU 可是很长、很长的），因此，内存容量不足将直接影响系统的整体性能。

对这一问题，显而易见的解决方法是增大物理内存空间，但这将增大系统成本。另一种方法就是在逻辑上扩充内存容量。“存储扩充”就是解决如何在逻辑上扩充内存容量，即虚拟存储技术。

虚拟存储器由内存和部分硬磁盘组成（图 3-6），目的是为了克服内存容量的局限性，实现在“小内存中运行大程序”。它力求将外存空间作为内存使用，在逻辑上实现内存空间的扩充，使用户在编程时只考虑逻辑地址空间，而不考虑实际内存的大小。

程序在虚拟存储器环境中运行时，并不是一次把全部程序都装入到内存，而是只将那些当前要运行的程序段装入内存，其余部分则存留在外存中。如果在程序的执行过程中所要访问的程序段尚未装入内存，则向 OS 发出请求，将它们调入到内存。如果此时内存已满，无法再装入新的程序段，则请求 OS 进行置换，将内存中暂时不用的程序段置换到外存去，腾出足够大的内存空间后，再将所要访问的程序段调入，使程序能够继续运行。

3.5.4 文件管理

文件是数据的一种组织形式，它可以是计算机处理的数据、图像、文本、程序等各类信息。由于内存的容量有限，且所存信息在断电后即丢失。因此，现代计算机系统中大量的程序和数据都是以文件的形式存放在外存中，需要时再调入内存。

如果由我们自己去管理外存上的文件，不仅需要了解文件在磁盘中是如何保存、如何提取、如何删除等的物理细节，而且在多用户环境下，要保证各用户在外存上存放的程序和数据不发生冲突、用户文件不被其他用户窃取和破坏，以及允许在一定条件下的多用户共享文件。显然，一般用户是很难胜任这些工作的，需要靠 OS 的文件管理程序来解决。

因此，文件管理的功能就是有效地管理文件的存储空间，合理地组织和管理文件，并为文件访问和文件保护提供更有效的方法及手段。具体如下：

- 解决如何组织和管理文件。若要创建一个新文件，用户只要指定文件类型和输入文件名（路径）即可。具体创建文件、分配存储空间等工作都是文件系统自动完成的。为了分配文件存储空间，文件系统首先对磁盘中存储块进行管理，包括建立空闲存储块表、对可用存储块进行分配，以及回收不用的存储块等。
- 实现文件的“按名存取”操作机制。用户按文件名进行操作，系统则是对文件实体进行操作。由文件系统自动完成由文件名到文件实体的对应操作。
- 提供文件共享功能及保护措施。
- 实现用户要求的各种操作，包括文件的创建、修改、复制、删除等。

1．文件的组织结构

计算机中存放着成千上万的文件，我们在使用计算机时，可以说随时都在对文件进行操作。我们通常看到的文件是按照目录来组织的，从最上层的根目录（如 C 盘盘符）到下一层子目录（文件夹），再到更下一层子目录，直到最底层的文件。这种提供给用户看的组织结构称为文件的逻辑结构（这是我们都比较熟悉的一种结构），而文件在存储设备上的存放形式称为文件的物理结构。

1）文件的逻辑结构

文件的逻辑结构是用户所观察到的文件组织形式，与物理存储介质无关。文件的逻辑结构分为有结构文件和无结构文件两种。在有结构文件中，文件由若干个相关记录组成（又称为记录式文件，如图 3-36 所示）。记录式文件由一条以上的记录（图 3-36 中的一行）构成，每个记录都包含若干的数据项（图 3-36 中的一列）。数据库文件就属于有结构文件。

SNO	SNAME	SEX	SCLASS	BDATE	RESUME
97000001	王小燕	男	软件971	1978-12-1	班长
97000002	刘丽华	女	软件972	1977-1-15	
97000003	秦刚	男	硬件971	1975-11-30	
97000004	李建国	男	硬件971	1976-6-24	
97000005	郝易平	女	软件971	1977-5-17	
97000006	杨双军	男	软件972	1978-4-28	
97000007	张清效	男	软件971	1979-1-23	副班长

7 共有记录数：10

图 3-36 记录式文件示意图

无结构文件可以看作是一个字符流（又称为流式文件），它由字符序列集合组成，其长

度以字节为单位。例如，一个源程序文件。可以把流式文件看作是记录式文件的一个特例。

文件系统设计的关键，就在于如何将记录构成一个文件及如何将一个文件存储到外存中，以便于系统对文件的快速检索。

2）文件的物理结构

文件的物理结构又称为文件的存储结构，是指文件在外存中的存储组织形式。计算机中最常用、基本的外存是硬磁盘。所以，文件的物理结构主要就是研究文件在硬磁盘上的存放和组织方式。通常将外存划分为若干个大小相等的物理块，相应地也将文件划分为相同大小的逻辑块，以块为单位进行存储和管理。需要指出的是，每个在逻辑上连续的各文件块，在外存上存放时却不一定能存放在连续的物理块中。

3）文件的目录结构

用户使用的是文件的逻辑结构，系统使用的是文件的物理结构，将这两种不同组织结构连接在一起的纽带就是文件的目录结构。

文件目录具有将文件名转换为该文件在外存的物理位置的功能。设计文件按目录组织的主要目的是实现对文件的“按名存取”，并提高文件的检索速度。

为了能对一个文件进行各种操作，文件系统为每个文件设置了一个包含该文件各种属性信息的数据结构，称为文件控制块（File Control Block，FCB）。文件和文件控制块一一对应，文件控制块的有序集合就称为文件目录，每个 PCB 就是一个文件目录项，文件目录本身也是文件，可以称为目录文件（你在 Windows 资源管理器中看到的目录文件就是一个个 FCB）。

文件的目录结构有单级结构、二级结构和多级结构。现代 OS 中都采用多级树形目录结构（图 3-37），目录的第一级称为根目录，目录树中的非叶结点均为子目录，树叶结点均为文件。

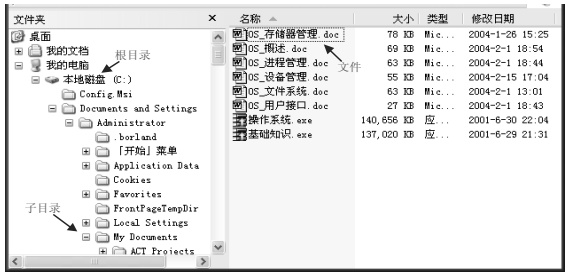


图 3-37　资源管理器窗口中显示的树形目录结构

2．文件存储空间管理

存储空间管理的主要任务是为文件在外存中分配必要的存储空间，并合理地组织文件的存取方式。

我们已经知道，硬磁盘是按磁道和扇区组织的。为了有效管理磁盘存储空间，通常将磁盘划分为大小相等的物理块，以物理块作为存储分配的基本单位。磁盘物理块以 2^n 个逻辑扇区（簇）构建（磁盘容量不同，n 的取值不同）。例如，将 1KB 或 512B 设为 1 个物理块。

常用的外存分配方法有以下几种：

- **连续分配**。将每一个逻辑文件中的记录，顺序存储到邻接的各物理盘块中，即为每一个文件分配一组相邻的物理块。如此，使文件中记录的逻辑顺序与其在存储器上的物理顺序一致。通常，地址连续的物理块都位于一条磁道上。读写时无需过多移

动磁头。因此，连续存放时对文件的访问速度比较快。缺点主要是需要连续的存储空间，这对大文件是比较困难的。

- **链接分配**。一个文件信息存放在非连续的物理块中[1]，各块之间通过指针连接，前一个物理块指向下一个物理块。为了能有效地管理和组织文件，OS 维护一个显示所有文件在硬盘中的起始扇区信息的表（当然，这张表本身也在硬盘中），找到了文件存放的起始扇区（文件第一块的存放处），就可以通过“链指针”的方式找到文件第二块的存放地址，依次，直到找到最后一块。这样，文件的所有物理块就被串联成为一个链表，块之间通过指针链接（图 3-38）。这里的“链指针”，就是在每个文件块的末尾处，会给出下一块文件的扇区地址。

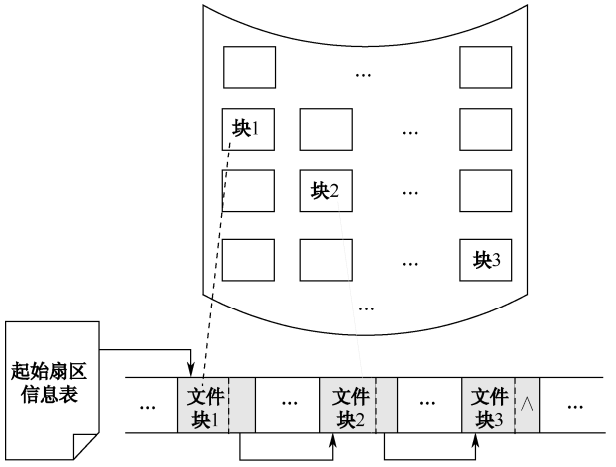


图 3-38　链接分配示意图

- **索引分配**。一个文件信息存放在非连续的物理块中。系统为每一个文件创建一张索引表，索引表的表项给出文件的逻辑块号和物理块号的对应关系。

除了为新创建的文件分配存储空间之外，操作系统还需要记住空闲存储空间（未用扇区）的情况。当用户要新建一个文件时，OS 会核查“空闲盘块表”，找到一个可以安放该文件的地方。如果用户要删除一个文件，OS 会同时更新文件在硬盘中的起始扇区信息表及空闲盘块表。首先删除第一张表中的该文件的入口信息，然后将这个文件腾出来的空间信息写入第二张表中。

文件的创建和删除动作贯穿于计算机的整个使用过程，这可能会造成空闲扇区零零散散地撒落在磁盘的不同角落（称为磁盘碎片），这会影响存储器的工作性能，特别是在数据查找时，会造成查询时间过长。因此，现代操作系统中都有磁盘碎片整理工具，可以将磁盘中的文件重新进行排列，使得磁盘上每个独立文件中文件块的存放扇区能够彼此尽可能地靠近，以提高磁盘的读写性能。

3．文件共享与保护

文件系统中存放着众多的文件，使人联想到如何对文件进行保护、免受无意或恶意的破

[1] 一个文件在磁盘中存放时，不一定都能够存放于一块连续的物理区域中。当一个文件大到不能在磁盘的一个物理块中放下时，就需要分成若干个“文件块”存放在不同的物理块中，而这些物理块可能是分散地处于磁盘中的不同地方。

坏？一个文件如何为多用户共享？这些都涉及到对访问文件的用户如何进行有效控制的问题。

文件共享是指允许多个用户或程序同时使用一个文件。一个文件能被多个用户共享最简单的方法就是凡需要使用该文件的用户都要自备该文件的副本，显然这会造成存储空间的极大浪费。无论是早期的文件共享方法还是现代文件共享方法，都是解决在一个文件副本的情况下多用户共享的技术和方法。不同的是，共享的范围不断扩大，从单机系统、多机系统、局域网系统，到现在的互联网范围中的文件共享。

文件保护实际上有两层含义：文件保护和文件保密。文件保护是指避免因有意或无意的误操作使文件受到破坏；文件保密是指未经授权不能访问文件。这两个问题都涉及到用户对文件的访问权限控制。

常见的文件访问控制方式有以下几种：

- **访问控制矩阵**。系统通过设置一个二维矩阵进行访问控制。二维矩阵的一维是所有用户，另一维是所有文件，对应的矩阵元素则是用户对文件的访问控制权，包括读（R）、写（W）和执行（X）权。当用户向文件系统提出访问请求时，由访问控制验证程序根据矩阵内容对本次访问请求进行比较，如果不匹配则拒绝执行。
- **访问控制表**。这种方法以文件为单位把用户划分成若干组，同时规定每组的访问权限。这样，所有用户组对文件访问权限的集合就形成了该文件的访问控制表。
- **用户权限表**。该方法是将一个用户或用户组所要访问的全部文件名集中存放在一个表中，且每个表项指明对相应文件的访问权限。
- **口令**。口令分两种：一种是系统使用权口令，用于验证是否有权进入系统；另一种是文件使用权口令，当任何用户想使用该文件时，都要核准口令后，才能访问操作。
- **密码**。密码方式在用户创建源文件并将数据写入存储设备时对文件进行编码加密，在读出文件时对其进行译码解密。只有能够进行译码解密的用户才能正确读出被加密的文件，从而起到文件保密的作用。

*3.5.5　其他功能

1．I/O 设备管理

设备管理主要是对计算机系统中的输入/输出设备进行分配、回收、调度和控制。其主要任务就是负责控制和操纵所有 I/O 设备，实现不同类型的 I/O 设备之间、I/O 设备与 CPU 之间、I/O 设备与通道和 I/O 设备与控制器之间的数据传输，使它们能协调地工作，为用户提供高效、便捷的 I/O 操作服务。

操作系统控制着系统中的所有基本 I/O 设备，禁止用户和应用程序直接处理这些 I/O 设备。例如，要读取磁盘中的某个文件，不需要给出文件在磁盘上的物理存放地址，而只需要简单地单击或者调用一个函数，具体到磁盘上去找文件的工作就由 OS 完成了。

2．用户接口

为了方便用户使用操作系统，OS 为计算机硬件和用户之间提供了交流的界面。用户通过 OS 告诉计算机执行什么操作，计算机系统为用户提供执行各种操作的服务，并按用户需要的形式返回操作结果。用户和计算机之间的这种交流构成完整的、人—机一体的系统，

将这个系统称为用户接口（Interface）。

随着 OS 功能不断的扩充和完善，用户接口更加人性化，呈现出更加友好的特性。目前，人—机之间的用户接口有两种主要类型：直接用户接口和间接用户接口。直接用户接口通过交互方式的用户界面进行人—机对话；间接用户接口通过程序命令进行系统调用的方式完成人—机交流，如图 3-39 所示。

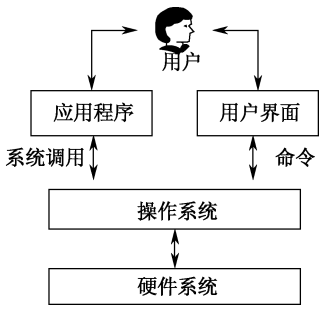


图 3-39　用户接口方式示意图

在计算机系统中，用户不能直接管理系统资源，所有资源的管理都是由 OS 统一负责的。但是，这并不是说用户就不能使用系统资源了，实际上用户可以通过系统调用的方式使用系统资源。这种在程序中实现的系统资源的使用方式被称为系统调用，或者称为应用编程接口 API。目前的 OS 都提供了功能丰富的系统调用功能。

不同 OS 所提供的系统调用功能有所不同。常见的系统调用分类如下：

- 文件管理。包括对文件的打开、读写、创建、复制、删除等操作。
- **进程管理**。包括进程的创建、执行、等待、调度、撤销等操作。
- **设备管理**。用于请求、启动、分配、运行、释放各种设备的操作。
- **进程通信**。用来在进程之间传递消息或信号等操作。
- **存储管理**。包括存储的分配、释放、存储空间的管理等操作。

3．系统启动

当我们希望运行一个应用程序时，操作系统会将它装载进内存。但操作系统本身是如何被装入内存并且执行的呢？处理上述过程的术语称为bootup（自举）。

计算机启动时，CPU 会使其内部的程序计数器 PC指针（见 3.3.2）指向一个特殊的位置，例如，Intel处理器会使PC指针指向内存（通常为 ROM）中地址为 0xFFFFFF0H 的地方。自举程序 boot loader 就存放在该地址为首的区域里。所以，一旦计算器启动，首先运行的就是 boot loader程序。

boot loader程序的目的是将OS从磁盘中装载进内存中，一种比较简单的方式是，boot loader程序会读取磁盘中的一段特定的区域，将该区域中的内容（就是OS）复制进内存中，然后在boot loader 程序的最后执行一条“无条件转移”指令，转移到该内存地址——OS便开始运行了。

一个典型的操作系统（如 Windows）会定义磁盘分区。例如，假设一块磁盘有1000个柱面，我们可以将前200个柱面作为一个分区，中间的 500 个柱面作为第二个分区，剩下的300柱面作为第三个分区。这些分区信息汇总为一张表，称为分区表，存放在物理磁盘的第一个块（Master Boot Record，MBR）中。

Intel 处理器将 boot loader 程序包含在 BIOS 中。当计算机启动时，BIOS 中的 boot loader 程序会读取分区表，将操作系统装入内存，然后如条件跳转到操作系统第一条指令的存放处，开始运行操作系统。

习题

一、填空题

1．微型计算机主机系统主要包括（　　）、（　　）、（　　）和（　　）等四个部分。

2．内存储器可分为（　　）和（　　）两类，其中，断电后所存信息就丢失的内存属于（　　）。

3．假设某内存储器有 1 K个单元，则至少需要（　　）bit 二进制码来表示该内存单元的地址。

4．每个内存单元中能存放（　　）bit 二进制数。

5．若已知磁头数为 10，柱面数为 4096，扇区数为 63 的硬盘容量为（　　）。

6．在计算机系统中设计 Cache 的主要目的是（　　）。

7．微型机中的存储器系统包括（　　）和（　　）两类。其中，（　　）的设计目标是提高存储器系统的存取速度。

8．计算机各部件传输信息的公共通路称为总线，一次传输信息的位数称为总线的（　　）。

9．PCIE 属于（　　）总线标准，而 SATA 则属于（　　）标准。

10．CPU 从外部设备输入或输出数据都需要通过（　　）。

11．计算机硬件能够直接识别的指令是（　　）。

12．冯•诺依曼计算机的基本原理是（　　）。

13．冯•诺依曼计算机结构是以（　　）为中心。

14．与冯•诺依曼结构相比，哈佛结构主要具有（　　）和（　　）两大特点。

15．某程序段经编译后生成 98 000 条机器指令，假设取指令、分析指令和执行指令所用的时间均为 2ns。则使用并行流水线方式完成该程序段所需的时间为（　　）ns。

16．如果说图灵机 A 能够完全模拟图灵机 B，则意味着（　　）。如果 A 和 B 能够相互模拟，则表示（　　）。

17．操作系统的基本功能包括（　　）、（　　）、（　　）、（　　）和用户接口。

18．进程在其生命周期中的三种基本状态是（　　）、（　　）和（　　）。

19．数据库中的文件类型属于（　　）文件。

20．程序装入内存时，源程序中的符号地址最终要变换为内存的（　　）地址。

二、简答题

1．试说明指令的执行步骤，哪些步骤是必须的？

2．简述冯•诺依曼计算机的特点。

3．简述进程和程序的区别。

4．说明为什么要引入进程。

5．将程序装入内存必须经过哪些步骤？

第 4 章　计算机网络及应用

引言

计算机网络是信息传输的基础设施，与信息获取、信息交换和信息发布密切相关。本章首先介绍了计算机网络的基础知识，然后介绍了互联网的一些基本概念和应用，最后简要描述了网络安全的概念、网络安全技术、常见的网络威胁及其防护技术。

教学目的

- 了解计算机网络的基本概念、分类和应用方式。
- 理解网络协议和网络体系结构。
- 了解互联网中 MAC 地址、IP 地址、域名等概念。
- 了解互联网接入技术。
- 了解常见的互联网的应用及其相关技术。
- 了解计算机网络安全及相关技术。
- 了解常见的网络威胁及其防护技术。

4.1　计算机网络基础

计算机网络是计算机技术与通信技术相结合的产物。如今，计算机网络在全球范围内得到了广泛的普及，日益深入到国民经济各部门和社会生活的各个方面，成为当今信息社会的重要基础和人们日常生活工作中不可缺少的工具。

4.1.1　计算机网络概述

1．计算机网络的概念

计算机网络（Computer Networks），是指将一些位于不同地理位置的、自治的计算机通过通信线路连接起来，实现资源共享和信息传输的计算机系统（图 4-1）。“自治”的计算机是指不依赖其他计算机而能够独立运行的“智能”系统，如微型计算机、智能手机等。计算机网络中的计算机又称为主机（Host）。

基于计算机网络的资源共享主要包括以下几种类型的资源：

- **硬件资源**。如大容量存储设备、网络打印机、绘图仪等大型设备。
- **计算资源**。如超级计算机系统提供的大数据量、高速运算能力。
- **软件资源**。如各种软件（包括共享软件和购买了许可证的收费软件）。

- **数据资源**。如各种统计数据、文献数据库、音视频、资料等。

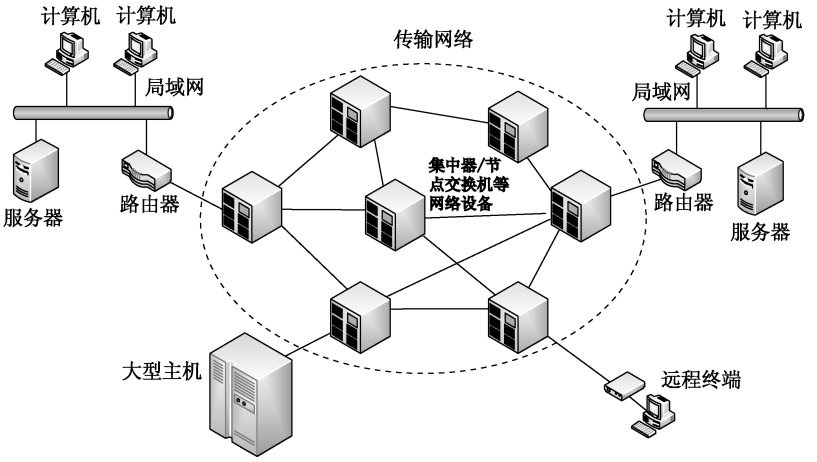


图 4-1　现代计算机网络示意图

在这些资源中，数据资源是最为重要的，因为其他几种资源可以通过购买而获得，而数据资源在很多情况下难以通过经济手段获得，特别是用户在工作中积累起来的数据。

最典型的计算机网络实例就是人们日常广泛应用的互联网（Internet）。互联网是世界范围内最大的计算机网络，它由成千上万个局域网（Local Area Network，LAN）和各种主机通过许多路由器（Router）互联而成，人们往往称其为“网络的网络”（Network of Networks）。

在概念上，与计算机网络类似的系统还有分布式计算机系统。分布式计算机系统可以在分布式操作系统支持下进行分布式数据处理和并行计算，也就是说，系统中的各计算机可以互相协调工作，共同完成一项任务，一个大型程序可以分布在多台计算机上并行运行。在分布式系统中，各计算机对用户是透明的。在用户看来，分布式系统就好像一台单独的计算机一样。分布式操作系统为用户选择一台或多台计算机来运行其程序，并将运行的结果合并传送给用户，这些都不需要用户的干预。分布式系统需要计算机网络作为底层通信支持。

2．计算机网络的发展

计算机网络源于计算机与通信技术的结合，其发展经历了如下几个阶段。

1）以单主机为中心的联机系统

以单主机为中心的联机网络系统被称为第一代网络。20 世纪 60 年代中期以前，计算机主机极为昂贵，而通信线路和通信设备的价格相对便宜，为了利用强大的主机处理能力来进行数据处理，人们通过通信线路将多台终端设备与主机连接，这样就构成了以主机为中心的联机网络系统，如图 4-2 所示。

在联机系统中，终端是没有处理能力的非智能设备（由键盘、显示器和通信接口构成），主机则具有强大的计算和处理能力。工作时，用户通过终端向主机发出操作请求，主机响应终端的请求完成相应的处理，并将结果回送给终端进行显示。

联机网络系统主要有三个缺点：一是主机负荷较重，需要承担数据处理和数据通信两方面的任务；二是通信线路的利用率低、成本高，尤其是距离较远时，分散的终端都要单独占用一条通信线路，不过，这个缺点可以通过在终端密集的区域采用终端集中器来合并

通信流量予以解决；三是联机系统属于集中控制方式，可靠性低，中央主机的失效直接导致整个系统的崩溃。

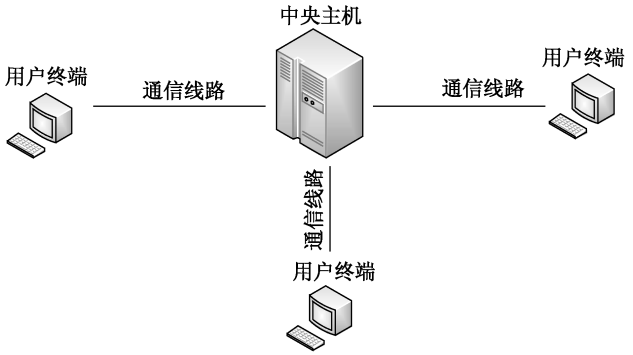


图 4-2　以单主机为中心的联机系统

2）分组交换网络的出现

随着计算机技术和通信技术的进步，从 20 世纪 60 年代中期开始，人们开始研究如何将多个联机系统互相连接，以便在更大范围内实现数据通信和资源共享。研究的结果最终导致了分组交换网络的出现。其间经历了两个演变过程，开始仅仅是简单地将各联机系统的主机通过通信线路互连，如图 4-3（a）所示。后来又将数据通信任务从主机中分离出来，交由专用的通信处理机（Communication Control Processor，CCP）完成，主机仅完成数据处理任务。CCP 之间通过通信线路互连，完成主机之间的数据通信。由多个 CCP 组成的网络称为通信子网，提供数据传输服务。而主机的集合称为资源子网，提供资源共享服务。两者构成了逻辑上具有两个层次的网络，如图 4-3（b）所示。

分组交换网的另一个特征就是将传输的数据分割成若干个分组进行传输，而不是像原来那样一次全部传输。这个改变是革命性的，它使网络的传输效率和可靠性有了极大的提升，为今天的 TCP/IP 网络打下了基础。

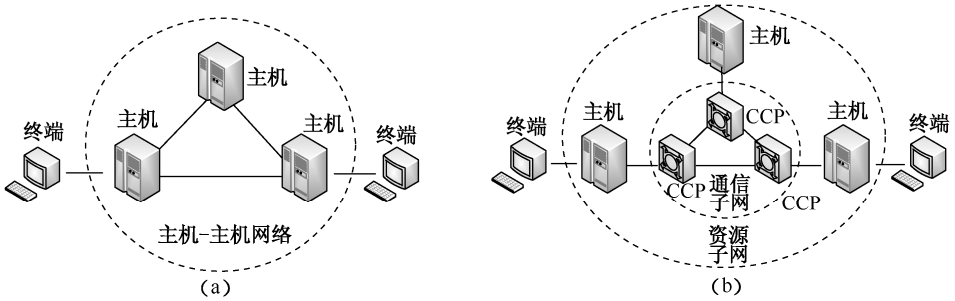


图 4-3　分组交换网的结构演变

3）网络体系结构标准化时代

20 世纪 70 年代以后，全世界出现了大量的计算机网络，它们大都由研究部门、大学或公司各自研制开发，没有统一的体系结构，难以实现互连。于是，开放（Open）就成了计算机网络发展的主题。

1977 年，国际标准化组织（International Standards Organization，ISO）开始研究计算机网络体系结构的标准化问题，经过多年艰苦的努力，于 1983 年推出了开放系统互连参考模

型（Open System Interconnection/Reference Model，OSI/RM），简称 OSI 标准。

目前，OSI 标准中所提出的关于计算机网络体系结构的概念已被人们广泛接受，成为网络研究和开发的“圣经”。也正是在 OSI 的推动和影响下，使得后来的 TCP/IP 体系结构迅速普及，TCP/IP 体系结构虽然不是国际标准，但它的发展和应用都远远超过了 OSI，并最终成为了互联网的标准。

4）互联网时代

20 世纪 90 年代，世界许多国家相继建立了本国的主干网络，并相互连接逐渐演变成了今天的互联网。互联网采用了 TCP/IP 协议标准，是一个由成千上万个网络构成的全球性的互联网。

3．计算机网络的组成要素

就像任何智能系统都是由硬件和软件两部分组成的一样，计算机网络同样由网络硬件和网络软件组成。

1）网络硬件

网络硬件主要分为两大类，即网络结点和通信链路。网络结点又分为端结点和转接结点。端结点是指通信的源设备和目的设备，例如，具有智能的主机系统和非智能的终端设备。转接结点是指用于在网络中控制和转发信息的中间结点设备，例如，各种交换机、集中器、多路复用器、调制解调器和光通信设备等。通信链路是指传输信息的信道，可以是有线的，如同轴电缆、光纤、双绞线等，也可以是无线的，如微波、无线电、红外线、卫星等。

2）网络软件

计算机网络是一个智能系统，需要通过软件来控制网络硬件的运行和管理。另外，为了安全有效地利用网络资源，为用户提供资源共享服务和信息发布服务，也需要通过软件对网络资源进行全面的管理、调度和分配，并提供安全防护功能。网络软件是实现网络功能不可缺少的软件环境。主要网络软件如下：

- **网络协议软件**。实现网络协议功能，通常被嵌入到操作系统中，如 TCP/IP。
- **网络应用客户端软件**。为用户提供使用网络的界面，如 IE 浏览器、Outlook、微信等。
- **网络操作系统**。实现系统资源共享、管理资源访问、提供网络通信的程序集合。流行的网络操作系统主要有 Windows、Linux 等。
- **网络工具软件**。实现对网络的监控、管理和维护等功能。

4．计算机网络分类

计算机网络的分类方法很多，从不同角度观察网络、分类网络，有利于全面了解网络系统的各种特性。常见的网络分类依据包括覆盖范围和拓扑结构。

1）按覆盖范围分类

（1）广域网（Wide Area Network，WAN）。

广域网的覆盖范围从几十公里到几千公里，可以是一个地区或一个国家，甚至是世界范围。广域网通常利用各种公用交换网络，将分布在不同地区的计算机网络或计算机系统互连起来。广域网的特点如下：

- 覆盖地理范围大。
- 适应大容量与突发性通信的要求。
- 适应综合业务服务的要求，提供 QoS 能力。
- 开放的设备接口与规范化的协议。
- 完善的通信服务与网络管理。

广域网的典型通信速率为 56Kb/s～622Mb/s。由于距离很远，广域网的传播延迟较大，可从几毫秒到几百毫秒（卫星信道）。

在拓扑结构上，广域网呈现为由许多交换机互联而成的网状结构，交换机之间采用点到点线路连接，采用的传输介质包括光纤、无线电、微波、卫星等。

在体系结构上，广域网工作在 OSI 参考模型的最低三层，主要采用存储转发方式进行数据交换。

（2）局域网（Local Area Network，LAN）。

局域网的覆盖范围通常为几米到几十公里，一般在一个建筑物内，或在一个工厂、一个企事业单位内部，为单位独自建设并自行管理。局域网的特点如下：

- 系统灵活性高，建设、维护、扩展、改造和升级都非常简单和容易。
- 传输速率高（10Mb/s～10Gb/s）。
- 传播延迟小，可靠性高。
- 使用成本低。

局域网的拓扑结构常见的是总线形和环形，这是由于其有限的地理范围决定的。局域网采用的传输介质包括双绞线、光纤和无线。

目前，非常流行的以太网（Ethernet）就是一种典型的局域网。

（3）城域网（Metropolitan Area Network，MAN）。

城域网的覆盖范围在广域网与局域网之间，其运行方式与局域网相似。如果不作严格的区分，城域网可以认为是一种城市范围的局域网，它可以覆盖一组邻近的公司或一个城市。城域网一般采用光纤作为传输介质，采用的技术既可以是广域网技术，也可以是局域网技术，可以支持数据、声音、图像和视频的传输，并有可能涉及当地的有线电视网。

2）按拓扑结构划分

拓扑（Topology）是从图论演变而来的，是一种研究与大小形状无关的点、线、面的特点的方法。在计算机网络技术中，抛开网络中的具体设备，把计算机、服务器、交换机、路由器等设备抽象为“点”，把通信介质或通信链路抽象为“线”，这样从拓扑学的观点来观察计算机网络系统，就形成了点和线组成的几何图形，从而抽象出了计算机网络的具体结构。这种采用拓扑学方法抽象的网络结构称为计算机网络的拓扑结构。

计算机网络的拓扑结构主要有总线型、星型、环形、树形、不规则形和全互连型等几种，如图 4-4 所示。网络拓扑结构对网络的设计、传输控制方法、传输技术、可靠性、费用等方面有着重要的影响。

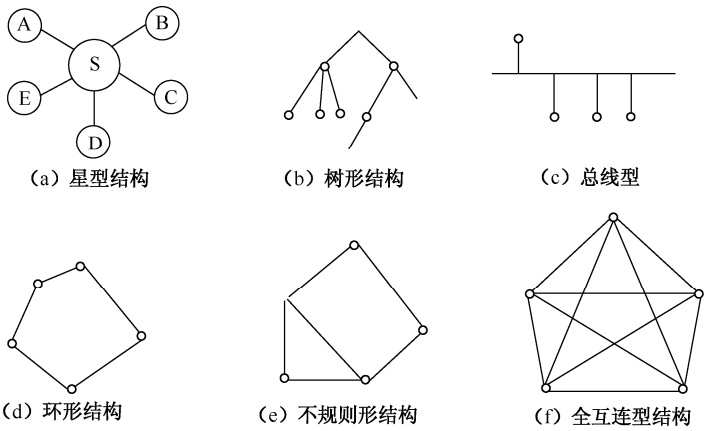


图 4-4　网络拓扑结构

（1）星型结构。

星型结构包括一个中心结点和一些通过点到点链路与中心结点相连的端结点（计算机）组成，如图 4-4（a）所示。中心结点有两种类型：一类是仅用来连接各种结点的集中连接设备，对传输的数据不进行处理；另一类是具有处理能力的设备，能够对传输的数据进行存储、处理和转发。星型结构的优点是构建容易，易于扩充，控制相对简单。其缺点是属集中控制，对中心结点的可靠性要求较高。

（2）树形结构（层次结构）。

网络呈现为一颗倒置的树形结构，树的叶子结点是计算机，根结点和中间结点是集中连接设备，如图 4-4（b）所示。树形结构如果仅有两级，就变为星型结构。一般来说，规模比较大的网络都采用树形结构来构建。

（3）总线型结构。

总线型结构是由一条公用总线连接若干个端结点所形成的网络，如图 4-4（c）所示。在总线型网络中，数据传输采用广播通信方式，即一个结点发送的信息会被广播到网络上的其他结点。由于总线是公用的，如果有两个以上的结点同时发送数据，就会产生冲突，因此，必须采取某种方法来控制链路的有序使用，这就是介质访问控制方法（Medium Access Control Method），其典型例子就是以太网中使用的载波监听多路访问/冲突检测（Carrier Sense Multiple Access/Collision Detect，CSMA/CD）。

总线型网络的优点是结构简单、成本低，缺点是结点发送数据时需要争用总线，所以实时性较差，当网络通信量增加时，性能会急剧下降。

（4）环形结构。

环形结构呈现为一种首尾相连的闭合环，环中的结点以点到点方式两两连接，如图 4-4（d）所示。环形网络常常使用令牌控制方法来协调各结点的数据传输。

（5）点—点部分连接的不规则形结构（网状结构）。

网络中的结点以点到点方式连接，每个结点至少有一条链路与其他结点连接，如图 4-4（e）所示。这种结构多用于广域网。广域网中各结点间的距离很远，某些结点之间是否使用点—点线路连接，要依据信息流量及结点的地理位置而定，如果结点间的通信可由其他结点转发而不影响传输性能时，则可不必直接互连。因此，网络覆盖地域范围很大且结点

数较多时，往往采用部分结点连接的不规则形拓扑结构。不规则形结构的网络必然会出现经由中间结点转发进行通信的现象，这称为交换。用于转发数据的设备称为交换机。

（6）点—点全互连结构。

如果网状结构中每个结点和其他所有结点都有连接，这就构成了全互连结构，如图 4-4（f）所示。全互连结构的连接数随结点数量的增加而迅速增加。例如，具有 10 个结点的全互连结构，每个结点需要 5 条线路与其他结点连接，则整个网络共需要 45 条线路。显然，在大型网络中，全互连结构的线路建设成本是非常高的。

5．计算机网络的应用

计算机网络刚出现时，它仅局限于一些大型企业、公司、校园与研究部门的特殊应用，但随着计算机网络的普及和计算机网络应用领域的扩展，计算机网络的应用已渗透到社会的每个角落和每个人的日常生活中。

计算机网络的应用与它所具有的功能是不可分的。计算机网络提供的基本功能有两个：资源共享和数据通信。以这两个基本功能为基础，可以衍生出大量各种各样的应用，应用举例如下。

1）企业信息化

机关、企业、校园内的计算机通常是分布在整幢办公大楼、工厂与校园内的。同时，有一些企业拥有多家工厂，这些工厂与公司的分支机构可能分布在世界各地。为了实现对全公司的生产经营情况与客户资料等信息的收集、分析和管理，可以将公司总部、厂区，以及分布在世界各地的分支机构办公室内的计算机连网，并构建以计算机网络为基础的企业信息系统，实现从产品设计、生产、库存、销售到财务、人事的全面管理。

2）社交网络

网络社会使人们的距离越来越近，人们可以通过电子邮件和即时通信（IM）与朋友进行随时随地的信息交流，通过 BBS、Blog、Twitter 等发布自己的话题和网络日志，通过网络花名册（Facebook）等建立社交网络形式的人际关系，通过 YouTube 发布自己的视频。

3）信息发布与共享

随着新闻走向在线与个人化，人们可以通过网络向公共传媒服务商订阅所感兴趣的在线新闻（或在线报刊杂志），然后服务商会将你订阅的内容传送到你的计算机或手机上。这种服务也可以应用于图书和学术论文的在线数字图书馆。

信息发布可以包括政府、教育、艺术、保健、娱乐、科学、体育、旅游等各个方面，甚至是各类的商业广告。这些信息可以通过门户网站和搜索引擎（如 Baidu、Google 等）进行检索和浏览。

4）电子商务

互联网在世界范围的普及和扩展，改变了商品的生产、销售和服务过程。万维网技术广泛应用于企业的业务流程，就形成了崭新的业务构架和交易模式，这就是电子商务。

针对企业而言，电子商务是基于计算机、网络通信等基础上的经济活动。它以互联网作为载体，使企业有效完成自身内部的各项经营活动，并解决企业之间的商业贸易和合作关系，发展和密切个体消费者与企业之间的联系，最终降低产、供、销的成本问题，增加企业利润，开辟新的市场。

针对个人而言，电子商务对人们的生活方式产生了深远影响。网上购物可以足不出门，购遍世界，网上银行可以方便地享受各种金融服务，使得消费者将能以一种十分轻松自由的自我服务方式来完成交易。

5）在线娱乐

在线娱乐正在对娱乐服务行业产生着巨大的影响。它可以让人们在家里点播电影和电视节目，收听网络电台和网络音乐，与他人玩网络在线游戏。

6）物联网

物联网能够把任何物品与互联网相连接，进行信息交换和通信，以实现智能化识别、定位、跟踪、监控和管理。它是集计算机网络、传感器技术、射频识别（RFID）技术和嵌入式技术为一体的综合性应用系统。

物联网的应用遍及智能交通、环境保护、智能小区、公共安全、智能家居、智能消防、工业监测、环境监测、老人护理、个人健康、农业园艺、水系监测、食品溯源、敌情侦查和情报搜集等众多领域。

4.1.2　计算机网络应用模式

网络应用通常有三种工作模式：C/S 模式、B/S 模式和 P2P 模式。

1．C/S（Client/Server）模式

即客户/服务器模式，按这种工作模式构建的计算机网络被相应地称为 C/S 结构的网络。C/S 模式的基本思想是将网络应用分解成多个子任务，分别由客户端和服务器来完成。通常情况下，客户端完成数据表示和用户接口任务，服务器端完成业务处理和数据库访问任务。

在基于 C/S 模式的系统中，客户端接收用户的输入，然后向服务器端发出数据请求，服务器端根据请求查找数据库，取出数据进行适当的处理，最后将结果传送到客户端，再由客户端对数据的表示形式和格式进行适当的转换呈现给用户。图 4-5 为 C/S 模式的示意图。

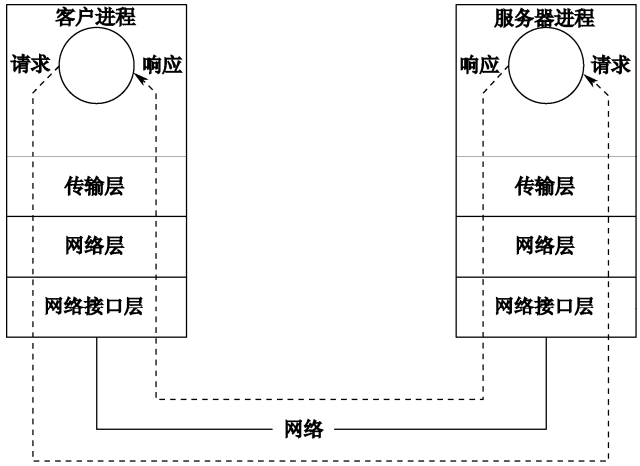


图 4-5　C/S 模式

2．B/S（Browser/Server）模式

即浏览器/服务器模式。B/S 模式是随着互联网和万维网（World Wide Web，WWW）

的发展而产生的。B/S 模式与 C/S 模式并没有本质的区别，B/S 模式可以认为是基于特定通信协议（HTTP）并采用网络浏览器作为客户端的 C/S 模式。它是为了满足瘦客户端（以网络浏览器作为客户端）的需要而产生的，可以大大节省客户端更新、维护等的成本。图 4-6 为 B/S 模式的示意图。

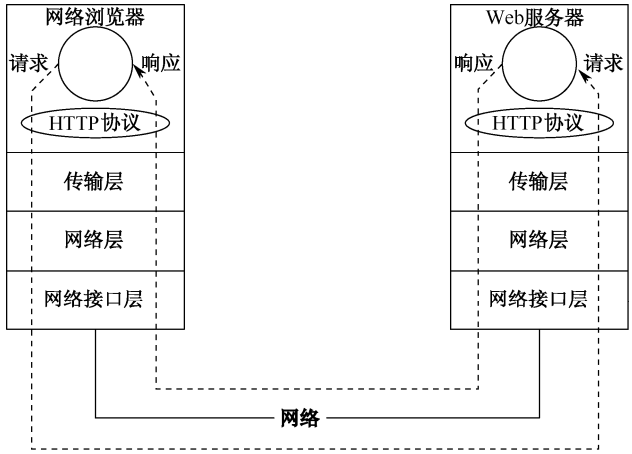


图 4-6　B/S 模式

3．P2P（Peer-to-Peer）模式

P2P 这个词有如下两个含义：

（1）一种网络组织方式。这时往往称按这种模式工作的网络为对等网络或工作组网络，意思是网络中的各计算机是按 P2P 模式工作的（当然也不排除其上还可以同时运行 C/S 应用或 B/S 应用）。

（2）一种网络应用工作模式。这时往往称按这种模式工作的应用为 P2P 应用或工作组模式的应用。

与 C/S 模式和 B/S 模式不同，C/S 模式和 B/S 模式都是以应用为核心的，在网络中必须有应用服务器，应用是通过用户的请求与应用服务器的响应来实现的，用户之间的通信也要在服务器的参与和控制下完成。而在 P2P 网络（应用）中，各方都具有同等的能力，它们既是客户端（可以发起一个会话请求），同时又是服务器端（可以响应一个会话请求）。用户之间可以直接通信、共享资源、协同工作，而无须设置专用的服务器。

4.1.3　计算机网络的体系结构和协议

计算机网络是一个复杂系统，涉及了众多的概念、原理、技术和方法。就像软件开发一样，人们在开发一个软件时往往会按照模块化、分层次（自顶向下）的方法来进行程序设计，在处理计算机网络这种复杂系统时也不例外，也采用了“分而治之”的层次化的方法对其进行研究、设计和开发。通过分层可以将庞大而复杂的问题转化为简单的局部问题，分层次处理和解决。

1．计算机网络体系结构

网络体系结构是用于设计和实现计算机网络的一组原则。它提供了独立于技术途径描

述计算机网络的相关概念，其中包括必须由网络执行的功能、信息格式与交互过程的描述等。

如上所述，为了降低研究计算机网络的复杂性，网络体系结构是按层（Layer）的方式来组织的，每一层的功能实现都以其下层为基础（换句话说，每一层的实现都要依赖于下层的服务），这些功能具体体现为一组通信协议，各层协议的集合称为协议集（或协议栈）。另外，为了实现各层的独立性，每层在功能上的具体实现细节对上一层屏蔽，层与层之间的耦合仅通过层间接口来实现。这就是计算机网络体系结构的分层思想。

简而言之，一个网络应包括哪些层次、各层具有哪些功能、各层包括哪些协议、层次间如何交换数据，这些都是网络体系结构所要研究的问题。

目前，最典型的计算机网络体系结构有两种：OSI/RM（7 层）和 TCP/IP（5 层）。

2．网络协议

在计算机网络中，通信双方要做到有条不紊地交换数据，就必须遵守一些预先约定好的规则，这些为进行网络中的数据交换而建立的规则、标准或约定就是网络协议。

网络协议的典型例子是互联网中使用的 TCP/IP 协议和局域网中使用的 CSMA/CD 协议。

典型的网络协议要包含三个方面的内容（三要素）：

- **语法**。数据与控制信息的组成结构或格式。
- **语义**。协议中控制信息的含义，如帧的起始定界符、源地址和目的地址、帧校验序列等。语义还规定通信双方对某种事件应该做出的动作及响应，如在什么条件下进行应答、什么条件下进行重发等。
- **时序**：规定了通信双方的操作顺序。例如，某协议规定：源站发送数据报文，若目的站接收的报文无差错，就向源站发送确认信息，通知源站报文已被正确接收；若目的站发现收到的报文有差错，就丢弃收到的报文；源站在规定时间内如果没有收到确认信息，则重发报文。

下面通过一个简单的例子来使读者对网络协议有一个感性认识。

例 4-1　网络中的站点 A 和站点 B 要进行点到点通信，为保证两个站点都能理解对方传送的信息并执行相应的操作，规定双方通信的协议如下：

（1）帧格式。

信息以帧为单位进行组织，帧格式为

SYN	STX	Text	ETX	FCS

其中：

SYN 标识一帧的开始，长度为 1B，编码为 16H；

STX 标识正文的开始，长度为 1B，编码为 02H；

Text 为信息正文，长度为 nB，允许的编码为 ASCII 字符集中的 20H～7EH；

ETX 标识正文的结束，长度为 1B，编码为 03H；

FCS 为校验码，长度为 2B，计算方法：从 SYN 到 ETX 所有字节之和（包括 SYN 和 ETX），高字节在前，低字节在后。

（2）传输规则。

以字节为单位进行发送/接收。

【发送方】

- 发送方按帧格式组帧。
- 将帧逐字节发送。
- 接收 ACK，若在 1s 内没有收到 ACK（06H），则重发此帧。若重发 5 次后还不成功，则报告错误。
- 结束。

【接收方】

- 检测 SYN 字符，若发现 SYN，则开始接收帧，直到接收到 ETX 为止，然后再接收 2B 的 FCS。
- 按 FCS 的计算方法求出一个 16 位的值，与接收到的 FCS 比较。若相同，说明接收的帧无差错，就发送 ACK（06H）进行确认；若不同，说明接收的帧有错误，就丢弃收到的帧，返回重试。
- 提取帧中的信息正文（Text 字段），保存到接收缓冲区中。
- 结束。

在这个协议例子中，所定义的信息格式就是协议的语法；帧中各语法元素的含义就是协议的语义；帧的传输规则就是协议的时序。一般情况下，本例中的帧统称为报文（Message），语法元素统称为字段（Field），其中的 Text 字段统称为数据（Data）或载荷（Load），Text 以外的其他字段统称为控制信息（Control Message）。要注意的是，不同协议的报文格式及报文中控制字段的个数和格式差别是非常大的。

3．协议分层与封装

根据网络体系结构中的分层原则，网络功能是被划分到不同层次实现的。或者说，通信过程中的各种问题在不同层次上予以解决的。

显然，通信双方要能够实现交互，双方同一层次（称为对等层）中的进程必须按照相同的规则来与对方通信，这就意味着双方同一层次的协议也应该是相同的。由此可知，网络的每一个对等层都需要有协议来规定双方在这一层次上的数据交换规则，即网络协议也是分层次的。概括地说：

- 网络体系结构中的每一层都有若干个通信协议，这些协议支配着本层通信双方之间的数据传输。
- 以网络体系结构的观点来观察网络中两个站点之间的通信，可以发现它是在多个对等层通信的支持下实现的。但对等层通信只是一种由软件实现的逻辑通信，实际的数据传输最终还是要归结为传输介质上的物理通信。

例 4-2 一个具有三层结构的网络中的数据传输过程。

在这个例子中，网络具有三层结构，如图 4-7 所示。假设网络中某一方的用户要发送数据给另一方的用户。

在发送方，用户数据首先提交给第 3 层，第 3 层会在用户数据的前面添加必要的控制信息 H3（又称为报文头部 Header），然后作为一个整体（称为协议数据单元，Protocol Data Unit，PDU）提交给第 2 层；第 2 层又会在第 3 层的 PDU 前面加上第 2 层的报文头部 H2

又作为第 2 层的 PDU 提交给第 1 层；同样，第 1 层会在第 2 层的 PDU 前面添加第 1 层的报文头部 H1，最后包含有第 1、2、3 层报文头部和用户数据的报文被放到物理介质上传送给对方。一般地，任何一个具有多层次结构的网络，发送方的每一层都会在上层提交的 PDU 前面添加本层的报文头部后再提交给下层。这种在数据前面加上报文头部的操作称为数据封装（Encapsulation），就像寄信前要将写好的信放入信封再邮寄一样。

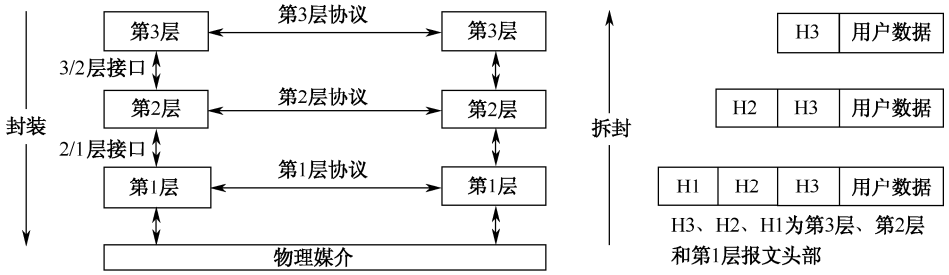


图 4-7　协议分层、封装和虚拟通信

在接收方，从物理介质上接收到数据后，首先提交给第 1 层，在该层去掉第 1 层的报文头部，然后向上提交给第 2 层，在第 2 层又去掉第 2 层的报文头部，然后向上提交给第 3 层，在第 3 层又去掉第 3 层的报文头部，最后提交给接收方用户（进程）。也就是说，接收方的每一层都会去掉 PDU 中本层的报文头部后再提交给上一层。这种去掉报文头部的操作称为数据解封装（Decapsulation）或拆封。

从这个例子可以看出，数据并不是在通信双方的对等层之间直接传输，而是在发送方和接收方相邻的两层之间依次传递。因此，对等层之间的通信只是一种通过协议实现虚拟通信。但正是在多层虚拟通信的支持下，数据才最终被传输到对方。

在封装过程中，不同的协议，报文头部包含的信息也各不相同。一般来说要包含源地址、目的地址、数据长度、传输控制信息、数据和错误校验信息等。

4．TCP/IP 协议及其体系结构

TCP/IP 是互联网所使用的标准协议。它实际上是一个协议簇，其中包含很多具体的协议，它们共同构成一个具有四个层次的体系结构，这四个层次从上到下分别是应用层、传输层、网际层和网络接口层，如图 4-8 所示。不过，其中的网络接口层并没有定义什么具体的内容，通常用 OSI/RM 中的数据链路层和物理层替代。

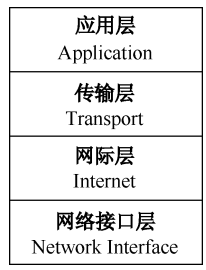


图 4-8　TCP/IP 体系结构

TCP/IP 体系结构各层的定义如下：

（1）应用层（Application Layer）为用户进程提供所需要的各种信息交换和远程操作服务。本层包含了很多面向应用的协议，如简单邮件传输协议（Simple Mail Transfer Protocol，SMTP）、超文本传输协议（Hyper Text Transfer Protocol，HTTP）、文件传输协议（File Transfer Protocol，FTP）等。本层的数据传输的单位（PDU）是报文。

（2）传输层（Transport Layer）又称为运输层，为应用层进程提供端到端（不涉及中间结点）的通信服务。本层定义了两个主要的传输协议：无连接的用户数据报协议（User

Datagram Protocol，UDP）和面向连接的传输控制协议（Transmission Control Protocol，TCP）。

UDP 在传送数据前不需要建立连接，如同邮递信件，属于发送后不管的方式，接收方接收到数据后也不需要给出任何确认，因此，UDP 提供的是一种不可靠的传输服务。但也正是由于不需要建立连接，因此，协议开销小，灵活、资源占用少。UDP 的数据传输的单位是用户数据报（User Datagram）。

TCP 提供面向连接的服务。如同打电话，在传送数据前必须先建立连接，数据传送结束后要释放连接。面向连接意味着 TCP 协议提供的是可靠的传输服务，但也意味着需要增加许多任务开销，如确认、流量控制、计时器及连接管理等。这不仅使报文头部增大很多，还要占用大量的处理机资源来处理这些任务。TCP 的数据传输的单位是报文分段（Segment）。

（3）网际层（Internet Layer），又称为互联网层或网络层，为网络上的不同主机提供通信服务，即解决主机到主机的通信问题。网际层提供的是一种无连接、不可靠但尽力而为的传输服务。本层提供的核心协议是互联网协议 IP（Internet Protocol），还包括一些辅助协议，如地址解析协议（Address Resolution Protocol，ARP）、互联网控制报文协议（Internet Control Message Protocol，ICMP）等。本层的数据传输的单位是数据报，数据报又称为分组或包（Packet）。

（4）网络接口层（Network Interface Layer）提供了将数据报通过物理网络传输的服务。在概念上本层对应于 OSI/RM 的数据链路层和物理层，但是 TCP/IP 并没有规定本层的协议，其目的是使 TCP/IP 能够适应不同的物理网络。在实际应用中往往会直接采用物理网络本身的相关协议。例如，局域网主要采用 IEEE802 系列协议，广域网常采用 HDLC、帧中继、PPP 等协议。

在 TCP/IP 网络中，物理网络是一个经常使用的概念。各种物理网络，如各种局域网、城域网、广域网甚至一条简单的物理线路之间的差异可能很大，但在 TCP/IP 看来，它们不过是主机之间的一条传输信息的“管道”而已。

4.2 互联网（Internet）

互联网是由成千上万不同类型、不同规模的计算机网络组成的世界范围的超大型计算机网络，使用 TCP/IP 协议作为其核心协议。

4.2.1 互联网基础

1．互联网的结构与组成

互联网的结构按逻辑功能和工作方式划分由两部分组成（图 4-9）。

边缘部分：由所有连接在互联网上的主机组成，是用户运行各种网络应用的终端设备。

核心部分：由大量的网络和连接这些网络的路由器组成，是为边缘部分提供数据传输服务的设施。

1）互联网的边缘部分

连接在互联网上的所有主机构成了互联网的边缘部分。这些主机可以是普通的 PC，或是智能手机或平板电脑，还可以是非常昂贵的服务器或巨型计算机。主机的拥有者可以是

个人，也可以是机构单位（学校、企业、政府机关等），当然也可以是某个 ISP（Internet Service Provider，互联网服务提供商）。边缘部分利用核心部分所提供的服务，使众多主机之间能够互相通信并交换或共享信息。互联网中的每个主机都有一个 IP 地址（更准确地说，主机中的每个网络接口都有一个 IP 地址）。

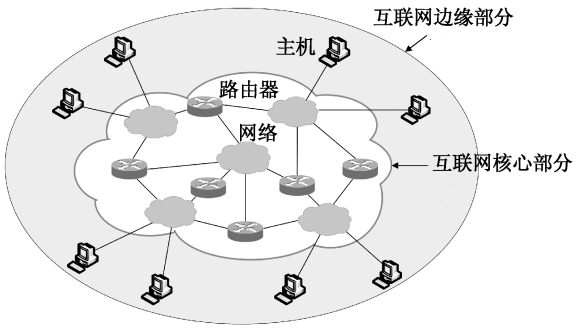


图 4-9　互联网的结构和组成

2）互联网的核心部分

互联网核心部分是互联网中最复杂的部分，核心部分要向互联网边缘中的大量主机提供连通性，使边缘部分中的任何一台主机都能够与其他主机进行通信。

互联网核心部分包含了大量的网络，这些网络可以是广域网，也可以是局域网，还可以是各种无线移动网络，所使用的数据通信技术也多种多样。所有的网络都通过一种称为路由器（Router）的设备连接在一起。每个网络都有一个唯一的网络地址（格式与 IP 地址相同）。

路由器工作在网络层，而网络层的数据传输的单位是分组，所以，互联网是建立在分组交换基础上的，这种以分组为单位进行传输的网络称为分组交换网（Packet Switching Network）。

从图 4-9 可以看出，一个网络的主机与另一个网络的主机之间的数据传输要跨越多个通过路由器连接的网络，而这之间的传输路径不止一条，因此，需要路由器选择一条最佳路径进行传输，选择最佳路径的策略和操作称为路由选择（Routing）。

路由器的工作原理是：路由器中建有一个存放到达其他网络的表格——路由表（Routing Table），当路由器从某个网络接收到一个分组时，首先提取出该分组的目的地址，然后根据目的地址查找路由表，找出应该从哪个网络将分组传送到下一个路由器，找到后就从连接该网络的接口转发该分组。路由表通常由路由器自动建立，建立路由表的算法称为路由算法（Routing Algorithm），而实现路由算法的协议称为路由协议（Routing Protocol）。

3）互联网接入

互联网核心部分所连接的网络可以是广域网，也可以是局域网。主机与互联网核心部分连接的接入技术根据主机的使用环境可以采用宽带接入、局域网接入或移动网络接入。

（1）宽带接入采用的技术属于广域网技术，典型的宽带接入技术有 ADSL、HFC 和 FTTx。

ADSL（Asymmetric Digital Subscriber Line，非对称数字用户线）接入主要用于家庭用户，ADSL 是一种上、下行传输速率不相等的传输技术。上行传输是从用户到 ISP 方向的传输，下行传输是从 ISP 到用户方向的传输。ADSL 的下行速率要远远大于上行速率，故

称为非对称数字用户线。ADSL 的技术特征如下（参考图 4-10）：

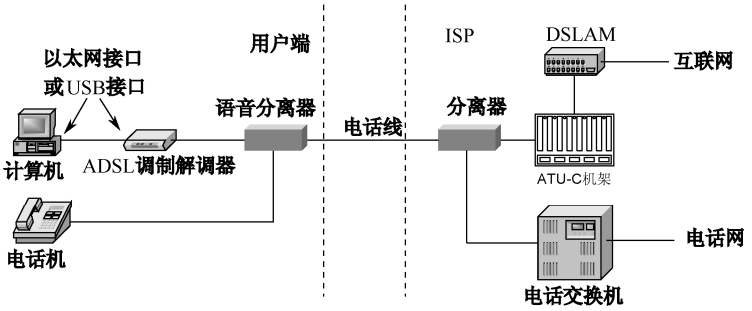


图 4-10　ADSL 接入配置图

- 使用普通电话线，无须另外架设线路。
- 能够在一条电话线上同时提供话音服务（打电话）和数据通信服务（上网）。
- 下行传输速率最高可达 8Mb/s，上行传输速率最高可达 1Mb/s，实际速率与用户到 ISP 之间的距离和电话线路质量有关。
- 用户端需安装用于上网的 ADSL 调制解调器和用于分离话音和数据的语音分离器。

HFC（Hybrid Fiber Coax）又称为光纤同轴混合网，它是在闭路电视网 CATV 的基础上开发的一种家庭宽带接入技术。除可传送电视节目外，HFC 网还提供电话、上网和其他宽带交互型业务。

HFC 的主要特点如下（图 4-11）：

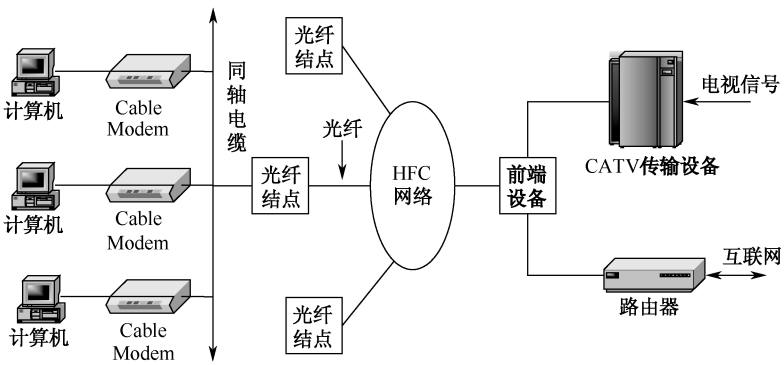


图 4-11　HFC 网接入配置图

- 主干线路采用光纤，并使用模拟光纤技术进行传输。分支结点到家庭的线路仍然使用 CATV 同轴电缆。
- 下行传输速率最高可达 30Mb/s，上行传输速率最高可达 10Mb/s。
- 采用介质共享方式使用上行信道。
- 用户端需安装用户接口盒和电缆调制解调器（Cable Modem）。用户接口盒用于连接电视机的机顶盒、电话机和电缆调制解调器。电缆调制解调器是用户主机通过 HFC 网络接入互联网的设备。

FTTx 是一种使用光纤的宽带接入技术。FTTx 中的字母 x 代表不同的地点。FTTx 的传

输速率可达 155Mb/s。

- 光纤到户 FTTH（Fiber To The Home）即将光纤一直铺设到用户家庭，这可能是家庭用户接入互联网的最佳方法。但由于费用问题和使用效率问题，目前将光纤铺设到每个家庭还无法全面普及。
- 光纤到楼 FTTB（Fiber To The Building）常用于一幢大楼的多个用户访问互联网。光纤进入大楼后就转换为电信号，然后用双绞线分配到各用户。这种方案可支持大中型企业、商业或大公司高速率的宽带业务需求。
- 光纤到路边 FTTC（Fiber To The Curb）实际上指的是光纤到居民区。光纤铺设到居民区后可为周边的一个或多个住宅小区的用户提供互联网接入服务。各个用户小区内部可使用局域网接入技术。

（2）局域网（以太网）接入。

由于以太网技术并不适用于远程通信，所以，严格意义上的以太网接入方式是不存在的。但如果不是很严格的话，则凡是用户通过单位或小区的以太网来访问互联网的方式也可以称为以太网接入。这时，以太网必须与其他宽带接入方式相结合。

（3）移动网络接入。

主要的移动网络接入技术包括 Wi-Fi 接入和电信网络的 2G/3G 接入技术。移动网络接入非常适合手机用户、平板电脑用户或无法使用有线连接的用户。

2G/3G 网络均采用蜂窝移动通信技术，2G 即第 2 代移动通信技术，包括 GSM（GRPS）和 CDMA 两种标准，数据传输率仅为 20～40Kb/s。3G 即第 3 代移动通信技术，包括 W—CDMA、CDMA2000 和 TD—SCDMA 三个标准，数据传输率可高达 21Mb/s。

Wi-Fi 网络是符合 802.11 标准的无线局域网（WLAN）。Wi-Fi 网络的覆盖范围较小，所以，仅在接入点（又称为热点，类似于手机通信中的基站）周围信号较好，连接速度较快。远离接入点时信号很差，可能会无法上网。

WiFi 接入的速度视接入点采用的 802.11 标准而定，802.11b 的数据传输速率为 11Mb/s，802.11g 的数据传输速率为 54Mb/s，802.11n 的数据传输速率为 100～300Mb/s。

2．互联网的数据传输过程

通过前面的叙述，已经知道互联网是一种采用 TCP/IP 协议的分组交换网，即用户数据被划分为小的分组在网上进行传送。下面以一份报文的传输为例介绍互联网中信息是如何传输的。注意，为了突出主线，此例中有些细节已进行了适当简化。

例 4-3　报文的传输过程。

本地主机要发送一份报文到远端主机时，TCP/IP 网络将会执行以下操作：

（1）应用进程将要发送的报文提交给传输层。

（2）传输层向远端主机发起 TCP 连接，连接使用四个参数来描述；本地主机 IP 地址、本地应用进程端口、远程主机 IP 地址、远端应用进程端口。端口标识了双方通信的进程是什么。

（3）TCP 连接建立起来后，TCP 把要传输的报文分解为多个报文分段，每个报文分段都封装上 TCP 头部（其中包括本地进程端口和远端进程端口），然后依次提交给网际层。

（4）在网际层，IP 协议将报文分段封装上 IP 头部（其中包括本地主机和远端主机的 IP 地址）形成 IP 分组，然后将分组发送到网络上。

（5）互联网中的一系列路由器都会按照分组中的目的主机 IP 地址寻找最佳传输路径，并依次从一个网络传送到另一个网络，最终送达远端主机所在的网络，并被网络中的目的主机所接收（目的主机会看到分组中的目的 IP 地址是自己）。

（6）收到的分组依次提交给服务器的网际层。

（7）网际层剥离掉分组前面的 IP 头部，将其中的报文分段提交给传输层。

（8）传输层剥离掉报文分段前面的 TCP 头部，放入缓冲区。当所有的报文分段都正确接收后，传输层将它们组装成报文提交给应用进程（TCP 知道报文分段中的端口号对应的是哪个应用进程）。

（9）双方断开 TCP 连接。

可以看出，以上报文的传输过程类似于日常生活中邮件的邮递过程，邮件要经过一系列封装和解封装：物品—邮件—邮包—集装箱—邮包—邮件—物品。并且在其投递过程中需要通过若干邮局的转发才最终到达收件人。

3．IP 地址和端口号

1）IP 地址

就像每个人都有一个居住地址一样，在互联网中，每个主机的每个网络接口都分配唯一的 IP 地址，它是互联网主机接口的“居住地址”。由于大多数情况下，每个主机只有一个网络接口，所以，可以认为网络接口的 IP 地址就是主机的 IP 地址。IP 地址可以让我们识别互联网中的不同主机。在 TCP/IP 体系结构中，IP 地址在网际层进行处理。

IP 地址用 32 位二进制编码表示。由于 32 位地址不容易记忆，所以，人们往往将 32 位的 IP 地址分为 4 个字节，每个字节间用圆点“.”分隔，并以等效的十进制数表示。这种表示法称为“点分十进制表示法”。因为一个字节所能表示的最小十进制数为 0，最大十进制数为 255，所以 IP 地址的范围在 0.0.0.0～255.255.255.255 之间。

例如，IP 地址：10000000.00000001.00000001.00000010，可表示为 128.1.1.2。

由于互联网中既有主机，也有大量的网络。为了使 IP 地址既能识别网络，又能识别主机，IP 地址又被分为网络地址（号）和主机地址（号）两部分。这就使得 IP 地址成为一种具有层次性的地址（居住地址也是有层次的，不过层次更多而已：省—市—区县—街道—小区—楼—室）。

IP 地址有 A、B、C、D、E 五大类，其中常用的是 A、B、C 三类，这三类地址的格式如图 4-12 所示（地址高位的几位固定编码用于区分地址类型）。

图 4-12　A 类、B 类、C 类 IP 地址的格式

A 类地址的网络号为 7 位，可提供使用的网络号是 126（2^7−2）。减 2 的原因是以下两个网络号不可分配给网络使用：

- 网络号为全 0 的 IP 地址是保留地址，意思是“本网络”。

- 网络号为 127（01111111）的 IP 地址保留作为本机软件回路测试（Loopback Test）之用。若主机发送一个网络号为 127 的 IP 分组，则自己将收到这个 IP 分组。

每一个具有 A 类地址的网络（简称 A 类网络）允许的最大主机数是 16 777 214（$2^{24}-2$），这里减 2 的原因是主机地址全 0 表示“本主机”，而全 1 表示“所有”，即该网络上的所有主机。A 类地址适用于拥有大量主机的大型网络。

B 类地址的网络号为 14 位，但规定 128.0.0.0 不能使用，所以，可供使用的网络号是 $2^{14}-1$（16 383）。B 类网络允许的最大主机数是 65534（$2^{16}-2$），减 2 的原因同上。B 类地址适用于中等规模的网络。

C 类地址的网络号为 21 位，但规定 192.0.0.0 不能使用，所以，可供使用的网络号是 $2^{21}-1$（2 097 151）。C 类网络允许的最大主机数是 254（$2^{8}-2$）。C 类地址适用于规模较小的局域网。

表 4-1 给出了保留的特殊 IP 地址，其用途也在表中进行了说明。

表 4-1　保留的特殊 IP 地址

网络号	主机号	源地址	目的地址	含义
0	0	可以使用	不可使用	本网络上的本主机
0	host-id	可以使用	不可使用	本网络上的某主机（host-id 指定）
全 1	全 1	不可使用	可以使用	在本网络中进行广播
net-id	全 1	不可使用	可以使用	在 net-id 网络上进行广播
127	全 0 或全 1 以外的任何数	可以使用	可以使用	本地环回地址

IP 地址中还有一些地址也是不能在互联网中使用的，这就是私有地址（Private Address）。私有地址是互联网规定专用于内网主机的 IP 地址。这里的内网是相对于公网（即互联网）而言的。内网是一些小型单位或家庭用户自己组建的局域网，其中的主机没有（或无法申请到）合法的 IP 地址，而私有地址就是专门为这类网络而保留的。私有地址不允许出现在互联网上，互联网中的路由器也不会转发那些具有私有地址的 IP 分组。

私有地址包括三个地址范围，这些地址都可以供内网自由选择使用：

- 10.0.0.0～10.255.255.255 （1 个 A 类地址块）。
- 172.16.0.0～172.31.255.255（15 个 B 类地址块）。
- 192.168.0.0～192.168.255.255（1 个 C 类地址块）。

使用私有地址的主机需要通过网络地址转换（Network Address Translation，NAT）才能访问互联网，NAT 可以将私有地址转化为互联网的合法地址。NAT 通常设置在内网的网关中。NAT 是解决 IPv4 地址不足问题的临时性的解决方案，它不适用于新的 IPv6 网络。

2）端口号

用 IP 地址可以标识一台主机，但如果一台主机中同时有多个进程进行传输，那么这些不同的进程应该如何标识呢？也就是说，主机接收到的数据应该传送给哪个进程呢？标识同一主机中不同进程的机制就是端口号（Port Number）。如果把 IP 地址看成是港口的地址，那么端口号就是该港口某个泊位的编号。

端口号是与传输层协议紧密相关的，因为传输层要通过端口与某个应用层进程打交道。传输层协议有两个（TCP 和 UDP），所以，端口号也分为 TCP 端口号和 UDP 端口号。意思是 TCP（或 UDP）通过哪个端口与相关的应用层协议打交道。

端口号是一个 16 位的正整数（0～65535），其中 0～1023 被固定分配给一些应用，这些端口号称为熟知端口号（Well-known Port Number）。49152～65535 为临时端口号，留给客户进程临时使用。临时端口号使用完后会被系统回收，可以再分配给其他客户进程重复利用。

一些常用的 TCP 熟知端口号所对应的应用层协议如下：

- 21 FTP（文件传输协议）。
- 23 Telnet（远程登录）。
- 25 SMTP（邮件传输协议）。
- 80 HTTP（超文本传输协议）。
- 110 POP3（邮局协议）。

4．子网和子网掩码

由上可知，32 位的 IP 地址所能表示的网络数是有限的。随着互联网中网络数量的增长，网络地址不够的问题日益严重。IPv4 的解决办法是采用子网寻址技术，将主机地址空间划出一定的位数作为网络号的一部分，划分出来的那几位就称为子网号，而剩余的主机地址空间作为子网的主机地址空间，这样一个网络就被分成多个子网。进行子网划分后，IP 地址就划分为“网络—子网—主机”三部分。

例 4-4　将 C 类网络 202.117.58.0 划分成 4 个大小相等的子网。

202.117.58.0 的二进制编码为 11001010.01110101.00111010.00000000，其中高 24 位为网络号，最低 8 位为主机号。划分 4 个子网需要从主机号部分拿出 2 位用作子网号，因此，4 个子网的网络地址和主机地址范围如下：

子网 1：11001010.01110101.00111010.00000000（202.117.58.0）；

子网 1 主机地址范围：202.117.58.1～202.117.58.63。

子网 2：11001010.01110101.00111010.01000000（202.117.58.64）；

子网 2 主机地址范围：202.117.58.65～202.117.58.127。

子网 3：11001010.01110101.00111010.10000000（202.117.58.128）；

子网 3 主机地址范围：202.117.58.129～202.117.58.191。

子网 4：11001010.01110101.00111010.11000000（202.117.58.192）；

子网 4 主机地址范围：202.117.58.193～202.117.58.254。

划分子网后，网络是如何识别各个子网的呢？区分一台主机属于哪个子网，可以通过子网掩码（Subnet Mask）。与 IP 地址相似，子网掩码也是一个 32 位的二进制数，也用点分十进制数表示。一个子网的子网掩码定义：对应于 IP 地址中的网络号和子网号部分，子网掩码中相应的位为“1”；对应于主机号部分，子网掩码中相应的位为“0”。

例 4-5　写出例 4-4 中子网的子网掩码。

按照子网掩码定义，例 4-4 中子网的子网掩码为

11111111.11111111.11111111.11000000（255.255.255.192）

对标准的 A、B、C 类网络，根据子网掩码的定义，可以知道它们的默认子网掩码为

A 类网络的默认子网掩码：255.0.0.0；

B 类网络的默认子网掩码：255.255.0.0；

C 类网络的默认子网掩码：255.255.255.0。

对于任何需要知道网络地址的操作，子网掩码都是不可缺少的参数：

- 网络可以根据子网掩码知道一个 IP 分组应投递到哪个子网中。
- 网关可以根据子网掩码知道是否应转发 IP 分组。
- 路由器可以根据子网掩码知道应通过哪个路径转发 IP 分组。

那么如何计算一个 IP 地址中的子网地址呢？方法很简单，只要将子网掩码与 IP 地址进行“与”运算，结果就是子网地址。根据子网地址的概念还可得出这样的结论：如果两个主机的 IP 地址经过子网掩码运算后结果相同，则表示两台主机处于同一网络内。

例 4-6　判断 IP 地址的子网属性。

（1）判断 C 类地址 202.117.35.239 和 202.117.58.114 是否属于同一子网；

（2）判断 202.117.35.239 和 202.117.35.200 是否属于同一子网，已知子网掩码为 255.255.255.192。

解

（1）首先将十进制表示的两个 C 类 IP 地址转换为二进制表示

202.117.35.239 的二进制表示：11001010.01110101.00100011.11101111；

202.117.58.114 的二进制表示：11001010.01110101.00111010.01110010。

C 类 IP 地址的默认子网掩码为 255.255.255.0，即

11111111.11111111.11111111.00000000

再分别用该子网掩码和两个 IP 地址进行逻辑“与”运算，运算结果为

202.117.35.239 的子网号为 202.117.35.0；

202.117.58.114 的子网号为 202.117.58.0。

两者的子网号不同，因此，这两个 IP 地址不属于同一子网。

（2）先将 IP 地址和子网掩码写成二进制。

202.117.35.239 的二进制表示：11001010.01110101.00100011.11101111；

202.117.35.193 的二进制表示：11001010.01110101.00100011.11000001；

掩码 255.255.255.192 的二进制表示：11111111.11111111.11111111.11000000。

再用子网掩码分别与两个 IP 地址进行逻辑“与”运算，运算结果为

202.117.35.239 的子网号为 202.117.35.192；

202.117.35.193 的子网号为 202.117.35.192。

两者的子网号相同，因此，这两个 IP 地址属于同一子网。

5．域名地址和 MAC 地址

在互联网中，除了 IP 地址和端口号，还有两类地址也是非常重要的：域名地址和 MAC 地址。

1）域名地址（简称域名）

由于数字表示的 IP 地址不便记忆，从 1985 年起，互联网在 IP 地址的基础上开始向用户提供域名到 IP 地址解析的域名服务（Domain Name Service，DNS），以方便用户使用便于记忆的域名来标识接入互联网的主机。域名有固定的格式，并通过域名服务与 IP 地址进行了绑定。例如，西安交通大学 Web 服务器的域名是 www.xjtu.edu.cn，经 DNS 解析后可知该域名对应的 IP 地址是 202.117.0.13。

DNS 系统由三个部分组成：域名空间、域名服务器和解析程序。DNS 将整个互联网视为一个域名空间，由不同层次的域（Domain）组成。每个域由自己的域名服务器来管理，在域名服务器中存放着主机的域名与 IP 地址之间的映射表。互联网中有许多域名服务器，分布在世界不同的地方，它们之间通过特定的协议进行相互通信和联系，这就保证了用户可以通过本地的域名服务器查找到互联网上的所有域名信息。

域名的结构由若干个分量组成，各分量之间用点“.”隔开。例如，一个具有三个层级的域名可以表示为

三级域名.二级域名.顶级域名

域名中的各分量分别代表不同层级的域名，层级最低的域名写在最左边（通常为主机名），层级最高的顶级域名则写在最右边。例如，mail.xjtu.edu.cn 表示西安交通大学的电子邮件服务器，其中，mail 为邮件服务器主机名，xjtu 为西安交通大学域名，edu 是教育科研域名，最右边的顶级域名 cn 为中国的国家域名。要注意，域名只是逻辑概念，并不反映主机所在的物理地点。

图 4-13 是 Internet 域名空间的树形结构，最上面的树根没有名字。根下面是顶级域结点，再下面是二级域结点，以此类推，最下面的叶子结点是主机名。一些常用的顶级域名的含义如表 4-2 所示。

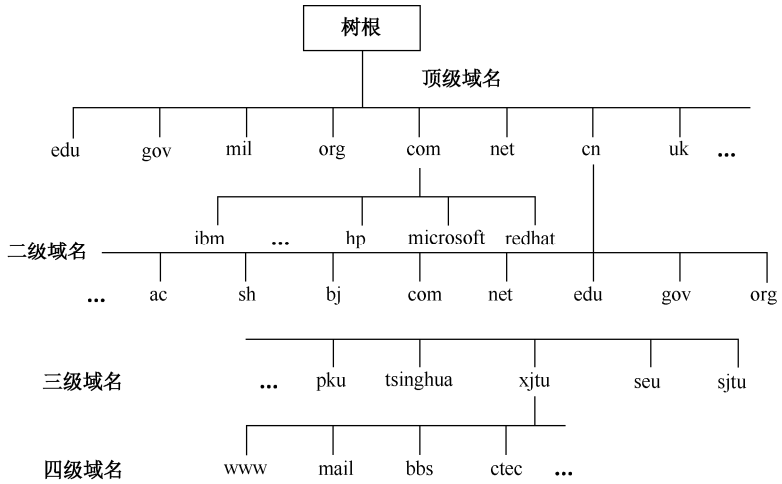


图 4-13　互联网的域名空间

表 4-2　常用顶级域名及其含义

顶级域名	含义
.edu	教育机构
.gov	政府部门
.mil	军事部门
.org	非营利性组织
.com	商业组织
.net	网络服务机构
.cn	中国

通过域名查找 IP 地址的过程称为域名解析（Domain Name Resolution）。域名解析由相关的域名服务器共同完成。域名解析基本过程如下：

当某个主机的应用进程需要通过域名访问另一个主机时，由于网络层只能识别 IP 地址，所以，必须先将目的主机的域名转换为对应的 IP 地址。应用进程首先将待转换的域名放在 DNS 请求报文中，发给本地域名服务器。本地域名服务器在自己的映射表中查找，如果没有找到，则将该 DNS 请求报文转发给顶级域名服务器，顶级域名服务器根据待查找的域名把 DNS 请求转发给相应的二级域名服务器，二级域名服务器再根据域名把 DNS 请求转发给相应的三级域名服务器，如此重复，当 DNS 请求送达负责管理该域名的域名服务器，该域名服务器在映射表中找到请求的域名对应的 IP 地址，然后将 IP 地址封装在 DNS 应答报文中，逐级回送，最终送达发出 DNS 请求的应用进程，然后应用进程就可以用 IP 地址和目的主机进行通信。

在实际应用中，为了减少域名查找的开销，减轻上面层级域名服务器的压力，每个域的域名服务器（包括本地主机）中都会设置一个 DNS 缓存，将相关的域名查询记录保存一段时间，这样，对于同样的查询只要在根据 DNS 缓存找到相关的 DNS 服务器直接查询即可。

例 4-7　利用 Windows 的 DNS 查询命令查询主机 www.baidu.com 的 IP 地址，然后清除本地 DNS 缓存。

Windows 的 DNS 查询命令为 nslookup，要执行题目指定的查询，可按下述步骤执行：

- 打开命令提示符窗口。
- 键入 nslookup www.baidu.com 回车。

执行结果如图 4-14 所示。

图 4-14　Windows 系统中的 DNS 查询命令

清除本地 DNS 缓存的命令是 ipconfig /flushdns。为了观察命令执行前后 DNS 缓存的变化，可按以下步骤执行：

- 执行 ipconfig /flushdns 清除 DNS 缓存。
- 执行 ipconfig/displaydns 查看 DNS 缓存，显示本地 DNS 缓存已经清空。
- 打开网络浏览器，访问www.baidu.com。
- 执行 ipconfig/displaydns，显示 www.baidu.com 相关的域名服务器记录。
- 执行 ipconfig /flushdns 清除 DNS 缓存。
- 执行 ipconfig/displaydns 查看 DNS 缓存，显示本地 DNS 缓存已经清空。

2）MAC 地址

MAC 地址是固化在网络接口硬件中的地址，又称为物理地址或硬件地址。它用于标识一个主机的网络接口。

MAC 地址由 48 位二进制编码构成（但以十六进制形式表示），如 00-15-58-EB-C1-DA。

MAC 地址和 IP 地址之间并没有必然的联系。MAC 地址就像一个人的身份证号，无论居住地在哪，身份证号永不会改变；IP 地址则如同一个人的居住地址（或邮政编码），搬家以后，其居住地址（或邮政编码）就随之发生改变。如果主机更换了一个新的网络接口，MAC 地址也随之改变，但主机的 IP 地址不会改变。反之，如果主机从一个网络移到另一个网络，其 IP 地址也要发生变化，但其 MAC 地址不会变化（因为网络接口没有更换）。

MAC 地址是网络接口层（更确切地说是数据链路层中的 MAC 子层）使用的地址。在数据链路层中，源主机要和目的主机进行通信，必须要知道目的主机的 MAC 地址，IP 地址是不能用来在数据链路层指定主机地址的。要将网际层的 IP 分组传送给目的主机，先要将 IP 分组封装到数据链路层的帧中才能发送给目的主机，而数据链路层的帧头部包含了通信双方的 MAC 地址。

如果想知道本机的 MAC 地址，在 Windows 系统下，打开命令提示符窗口，输入 ipconfig /all，就可以看到本机的 MAC 地址了，如图 4-15 所示。注意，此命令也会显示本地主机的 IP 配置信息。

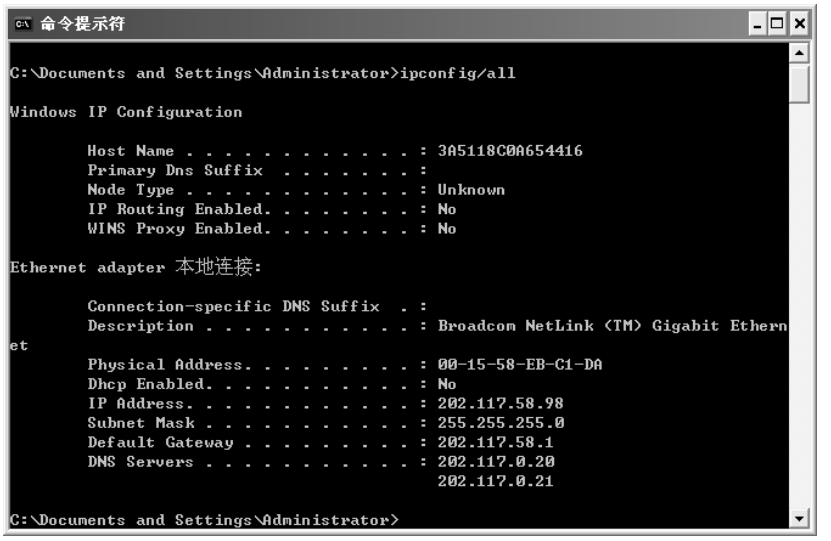


图 4-15　使用 Windows 的 ipconfig 命令查看本机的网络配置

6．局域网

大多数人初次上网接触到的都是局域网，大多数互联网中的主机也都是通过局域网接入的，特别是企业、政府部门、学校和研究机构等企事业单位中的主机。即便是家庭用户用宽带上网也不可避免地涉及到局域网技术。

从网络体系结构上观察，局域网技术只涉及 OSI/RM 定义的 7 层中的最低两层，即数据链路层和物理层。在局域网中，当一个 IP 分组由网络层传递到数据链路层时，IP 数据包将被封装在数据链路层的 PDU——帧（Frame）中，然后送到物理层进行传输。反之，当

物理层从传输介质上接收到一帧时，它就将这个帧提交给数据链路层，然后由数据链路层从帧中剥离出 IP 分组提交给网络层进行处理。

局域网的外部特征表现为较小的覆盖范围（20km 以内）、极高的传输速率（桌面连接可达 1Gb/s，主干连接可达 10Gb/s）和自建自管。

典型的局域网是以太网（Ethernet）和 802.3 局域网，这两种局域网几乎没有什么区别，对一般用户来说可以等同看待。

1）局域网的组成结构

局域网的组成较为简单，其主要部件如下：

- 计算机。
- 网络设备（网络接口卡、交换机、路由器/网关、无线接入点等）。
- 传输介质（双绞线、光纤、无线介质等）。

图 4-16 是一个简单的局域网组成示意图。

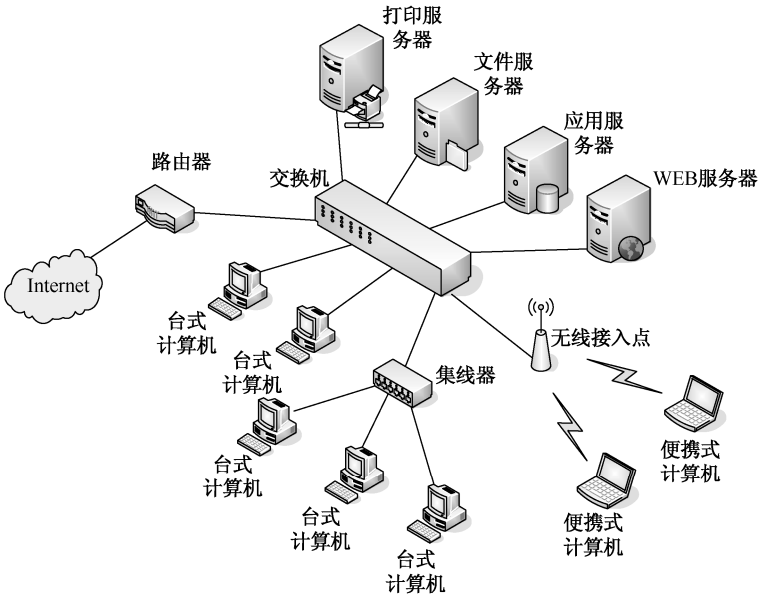


图 4-16　一个简单的局域网组成示意图

2）局域网的技术特征

局域网的主要技术特征有三个：网络拓扑结构、信号传输形式和介质访问控制方法。

（1）网络拓扑结构。

局域网常见的拓扑结构有星（树）形、总线形和环形。

星形拓扑结构在现代局域网应用较多，特别是集线器（Hub）和交换机（Switch）在局域网中的大量使用，使得星形结构成为局域网的主流结构。总线拓扑也是局域网中非常流行的一种拓扑形式，具有可靠性高，扩充方便的优点，曾广泛应用于著名的以太网（Ethernet）中，但目前已被星形结构所替代。环形结构只出现在早期的令牌环网中，现在已不多见。

（2）信号传输形式。

局域网的信号传输形式有两种：基带（Base Band）传输与宽带（Broad Band）传输。基带传输使用的典型传输介质有双绞线、同轴电缆和光纤，宽带传输使用的典型传输介质

有同轴电缆、光纤、微波和无线介质等。基带传输主要用于有线局域网，宽带传输主要用于无线局域网（Wireless LAN，WLAN）。

（3）介质访问控制方法。

介质访问控制方法是指网络中的多个站点如何共享信道。由于历史原因，局域网的通信采用了总线形拓扑结构上的广播传输方式，局域网采用了随机接入的多点访问型介质访问控制方法 CSMA/CD，随着无线局域网的问世和快速普及，CSMA/CA 也成为局域网的介质访问控制方法之一。

3）局域网的标准

IEEE 802 系列标准（由 IEEE 802 委员会制定的局域网标准，802 标准）是最成熟、标准化程度最高的局域网标准。802 标准不仅已被美国国家标准局 ANSI 接纳为美国国家标准，还被 ISO 正式接纳为国际标准（ISO 8802），从而得到了世界各国广泛的应用。IEEE 802 系列标准覆盖了传输介质、介质访问控制方法、组网方式，以及网络测试和管理的内容。

802 标准完全遵循了 OSI 参考模型的原则。它主要描述了网络体系结构中最低两层——物理层和数据链路层的功能。

802 标准的物理层规定了物理传输所使用的信号编码和介质，规定了网络的拓扑结构和传输速率。802 标准的数据链路层分为逻辑链路控制（Logical Link Control，LLC）和介质访问控制（Media Access Control ，MAC）两个功能子层。这两个子层将数据链路功能中与硬件相关的部分和与硬件无关的部分分离开来，从而使局域网体系结构在 LLC 子层不变的条件下，只需更换 MAC 子层便可适应不同的传输介质和介质访问控制方法。这是网络体系结构分层思想在局域网中成功应用的一个典型案例。

802 标准由一系列的子标准组成，并且还在不断扩充。其中主要的子标准如下：

- IEEE 802.2 逻辑链路控制。
- IEEE 802.3 CSMA/CD 总线访问控制及物理层规范（以太网）。
- IEEE 802.11 无线局域网访问控制及物理层规范（WLAN）。

4）局域网的介质访问控制方法

有线局域网的介质访问控制方法主要以 CSMA/CD 为主。CSMA/CD 是一种采用随机访问技术的竞争型（有冲突的）介质访问控制方法。采用 CSMA/CD 的网络中，站点必须具有判断信道忙（闲）的能力，判断的方法是，每个站点的接收器需要监听传输介质，如果介质上有信号传输，说明介质正在被某站点使用（介质忙）；如果介质上没有信号，则介质空闲。

当一个站点要发送数据时，就启动 CSMA/CD 过程，规则如下：

- 如果介质空闲，则开始发送。
- 如果介质忙，则继续监听，一旦介质空闲，再等待一小段时间然后返回重试。
- 发送时需要继续监听介质，若发送期间检测到冲突，就立即停止发送，并向介质发送一串阻塞信号以强化冲突。
- 发送阻塞信号后，等待一段随机时间，返回重试。

从 CSMA/CD 规则可以看出，如果发送时产生冲突，所有发送站都将停止发送，然后按规则重新竞争介质的使用权，这样就不至于在产生冲突后还继续将剩余的数据发送完，使这段时间大家都无法占用介质。

5）局域网设备

构建局域网所需的设备主要有网络接口卡、网络交换机和网关，如果要构建无线局域网，则需要无线网卡、无线接入点（Access Point，AP）或无线路由器（AP、交换机和路由器三合一设备）。

（1）网络接口卡（Network Interface Card，NIC）。

网络接口卡简称网卡，又称网络适配器。早期的网卡是插在计算机总线插槽内或 USB 接口上的扩展卡，称为独立网卡。现在的计算机主板上大多数已经集成了网络接口，称为集成网卡。在局域网中，每一台计算机都必须有至少一个网络接口。

（2）网络交换机（图 4-17）。

网络交换机是一种集中连接设备，利用它可轻松地组建以双绞线为介质的星形结构局域网。交换机工作在数据链路层，能够根据 MAC 地址将帧转发到合适的连接端口。

用网络交换机组网还有一个很大的优点，交换机的每个接口都是一个独立的冲突域，因此，它能够隔离冲突域，减低接口所连接之网段的冲突概率。

图 4-17　网络交换机（Netgear 12-port 10/100/1000Mbps Managed Fiber Gigabit Switch）

（3）网关（Gateway）。

局域网中的网关实质上是一台路由器，是局域网通向互联网的关口。当局域网中的主机要与互联网中的其他主机进行通信时，必须要通过网关。图 4-16 中与互联网相连的路由器就是该局域网的网关。

局域网可以设置多个网关，主机可以指定 IP 分组从哪个网关发到互联网。如果局域网中只有一个网关，则可以在主机中将该网关地址设置为默认网关。默认网关的意思是主机如果找不到可用的网关，就把 IP 分组发往默认网关，由默认网关来转发 IP 分组。在主机中，默认网关是不能随便指定的，如果设置不正确，IP 分组就会传送到不是网关的主机，从而无法访问互联网。ARP 病毒就会修改主机上的网关设置，造成不能上网的现象。

（4）无线接入点。

无线接入点是无线局域网中的“无线基站”，类似于有线局域网中的集线器，用于为无线站点之间或无线站点与有线局域网之间提供通信转接能力。在一个具有 AP 的 WLAN 中，无线站点可以直接与另一个无线站点通信，也可以通过 AP 与另一个无线站点通信（此时 AP 的作用是负责站点之间的信息转发）。如果把无线接入点 AP 与有线局域网连接，它还可以用作 WLAN 和有线局域网之间的桥接器，将多个无线站点接入到有线局域网上。

在构建家庭网络和 SOHO（Small Office & Home Office）网络时，还经常用到一种称为无线路由器（Wireless Router）的设备（图 4-18）。无线路由器将广域网接口、无线网络接口和以太网交换机集成在一起，既有路由器的功能，又有 AP 的功能，还有网络交换机的功能（通常有 4 个有线以太网接口），这就为构建小型有线/无线混合型网络带来了极大的方便。在网络的逻辑结构上，通常把无线路由器当作 AP 看待。

图 4-18　无线路由器（AP+路由器+交换机）

4.2.2　互联网应用

本节将介绍计算机网络的通信服务是如何提供给应用进程来使用的，换句话说，各种应用进程通过什么样的应用层协议来使用网络所提供的通信服务。

应用层的许多协议都是基于 C/S 方式。这里再强调一下，客户和服务器都是指通信中所涉及的两个应用进程。C/S 方式所描述的是进程之间服务和被服务的关系。这里最主要的特征就是客户是服务请求方，服务器是服务提供方。

本节讨论的内容包括电子邮件、万维网和文件传输。

1．电子邮件

电子邮件（E-mail）是 Internet 上最基本、最常用的一种应用，它不仅可以传送邮件本身，还可以以附件形式传送文字、声音、图像、数值数据等内容。电子邮件地址的格式：用户名@用户邮箱宿主主机的域名。

一个电子邮件系统主要由用户代理、邮件协议和邮件服务器三个部分组成。

用户代理（User Agent）是用户和电子邮件系统的接口，为用户提供一个友好的发送和接收邮件的界面。用户代理软件有很多，如 Windows 平台上的 Outlook、Foxmail 等。

邮件服务器需要使用两种不同的协议：

- SMTP 协议（简单邮件传输协议）用于用户代理向邮件服务器发送邮件和在邮件服务器之间传送邮件，它是电子邮件系统中邮件传输的标准应用协议，借助于传输层的 TCP 协议进行信息传输。两台使用 SMTP 协议的计算机通过 Internet 实现了连接，它们之间便可以进行邮件交换。
- 邮局协议 POP3（Post Office Protocol version 3）用于用户代理从邮件服务器读取邮件。在电子邮件系统中，用于存储和投递 Internet 电子邮件的主机称为 POP3 服务器。POP3 服务器与邮件服务器通常位于同一台主机。

邮件服务器是电子邮件系统的核心构件，其功能是发送和接收邮件，同时还要向发信人报告邮件传送的情况。邮件服务器按照 C/S 方式工作，它兼有客户和服务器两种角色，当它向目的邮件服务器发送邮件时，它就是 SMTP 客户；当它从用户代理接收邮件或从源邮件服务器接收邮件时，它就是 SMTP 服务器。因此，邮件服务器内部始终运行着两个进程，一个是 SMTP 客户进程，一个是 SMTP 服务器进程。

电子邮件发送和接收的过程（图 4-19）如下：

- 发件人使用用户代理编写信件，然后用户代理向发送方邮件服务器发起 TCP 连接请求。

- 当 TCP 连接建立后，用户代理使用 SMTP 协议将邮件发送给发送方邮件服务器，然后关闭 TCP 连接。
- 发送方邮件服务器将邮件放入邮件发送队列中，等待发送。
- 当发送方 SMTP 客户进程发现邮件发送队列中有邮件时，就与接收方邮件服务器建立 TCP 连接。
- 当 TCP 连接建立后，发送方邮件服务器的 SMTP 客户将邮件传送给接收方邮件服务器的 SMTP 服务器，然后关闭 TCP 连接。
- 接收方邮件服务器的 SMTP 服务器将接收到的邮件放入收件人的邮箱中（实际是一个用户目录），等待收件人方便时读取。
- 收件人收信时，运行用户代理，用户代理与接收方邮件服务器建立 TCP 连接。
- 当 TCP 连接建立后，用户代理使用 POP3 协议将收件人的邮件从接收方邮件服务器中的邮箱中取回，然后关闭 TCP 连接。

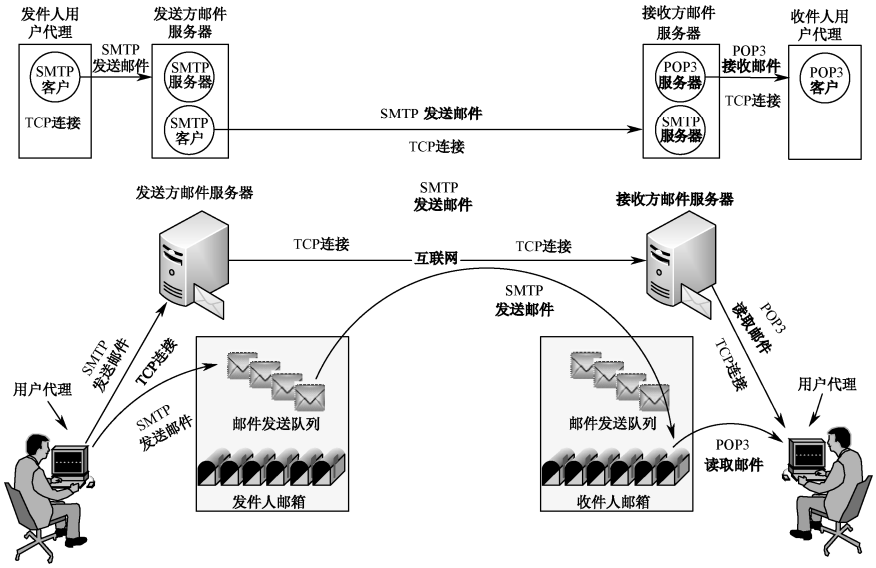


图 4-19　电子邮件系统的组成及工作原理

2．万维网

1）万维网的概念

万维网（World Wide Web，WWW）又称为 Web、3W 等。万维网使用超文本（Hypertext）来组织、查找和表示信息，并利用超链接（Hyperlink）将互联网中的相关资源链接起来，构成了一种分布式的资源共享服务，使用户能够按需获取信息，这样就彻底摆脱了以前查询工具只能按特定路径一步一步地查找信息的限制。图 4-20 给出了万维网的逻辑视图，图 4-20 中也说明了万维网与互联网、广域网之间的关系。

在逻辑上，万维网包含三个组成要素：浏览器（Browser）、Web 服务器（Web Server）和超文本传送协议（HyperText Transfer Protocol，HTTP）。

浏览器在万维网上是与 Web 服务器打交道的客户端程序。浏览器能够按照规定的格式显示 Web 服务器或文件系统内的文档。常见的浏览器有 IE、Chrome、Firefox、Opera 等。

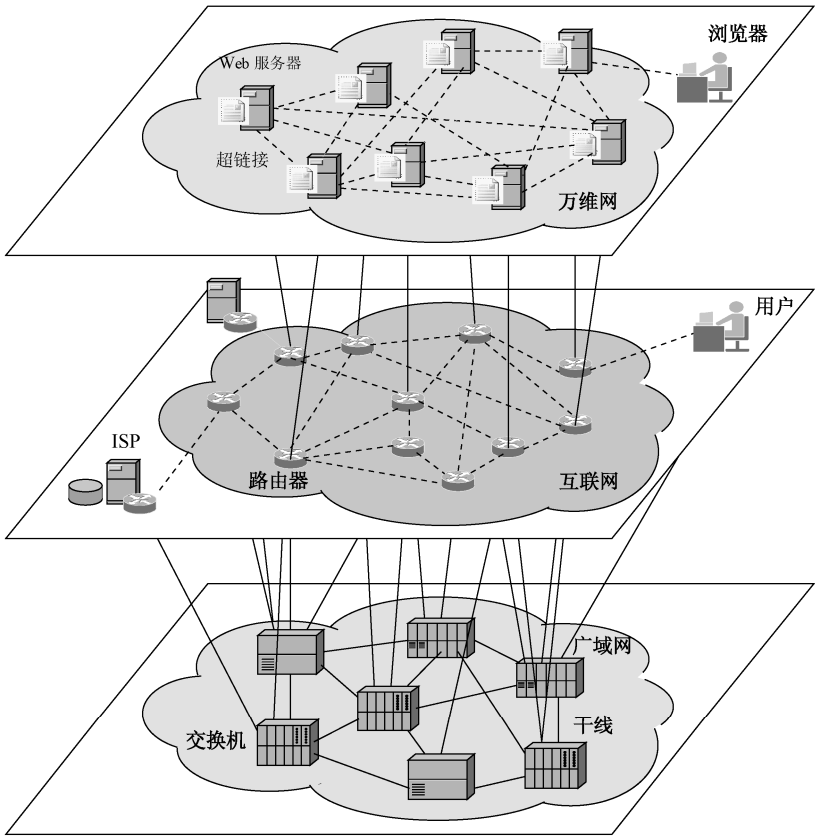


图 4-19　万维网与互联网、广域网之间的关系

Web 服务器是万维网上提供 Web 服务的程序。当浏览器连接到 Web 服务器并请求某个网页文件时，Web 服务器将根据该请求将网页文件传送给浏览器。Web 服务器使用 HTTP 协议与浏览器进行信息交互，所以它又称为 HTTP 服务器。Web 服务器不仅能够存储信息，还能根据浏览器提供的信息运行脚本（Script）和程序。常见的 Web 服务器有 Apache 和 Microsoft 的 IIS（Internet Information Server）。Web 服务器必须运行在一台互联网主机上。

2）网站、网页与 HTML 语言

万维网是一个分布式信息系统，是由大量的网页（Web Pages）组成的。网页文件由超文本标记语言（Hypertext Markup Language，HTML）编写，内容包括文字、图片、动画、声音等多种媒体信息，以及实现与其他网页、网站或资源的关联和跳转的超链接（Hyperlink）。网页能被浏览器识别、解释并显示，它本身是一个文本文件，扩展名为.htm 或.html。

具有特定主题的相关网页的集合称为 Web 网站（Website）。网站建立在 Web 服务器上，一个网站上有多少网页没有明确的规定，即使只有一个网页也能称为网站。进入 Web 网站看到的第一个网页称为首页或主页（Homepage）。

要制作一个网站，首先需要单独编辑若干个网页文件，然后通过超链接建立起它们之间的逻辑连接关系，并存入 Web 服务器的发布目录，这样网站就建好了。

要访问一个网站，只需要在浏览器的地址栏中输入该网站所驻留的 Web 服务器主机的 IP 地址或域名即可。

例 4-8　编辑一个简单的网页文档并在浏览器中显示。

在 Windows 中打开记事本，键入以下内容，然后另存为 index.html 文件。双击该文件即可启动浏览器将网页显示在屏幕上。显示的结果如图 4-21 所示。

```
<HTML>
  <HEAD>
    <TITLE>一个简单的网页文档例子</TITLE>
  </HEAD>
  <BODY>
   <H1>这是一个简单的网页文档例子</H1>
   <P>建立：此文件在 Windows 中用记事本建立，另存为 index.html 文件。</P>
   <P>执行：双击 index.html 文件。</P>
   <P>单击<A href="http://www.baidu.com/">这个链接</A>会打开百度的主页。</P>
  </BODY>
</HTML>
```

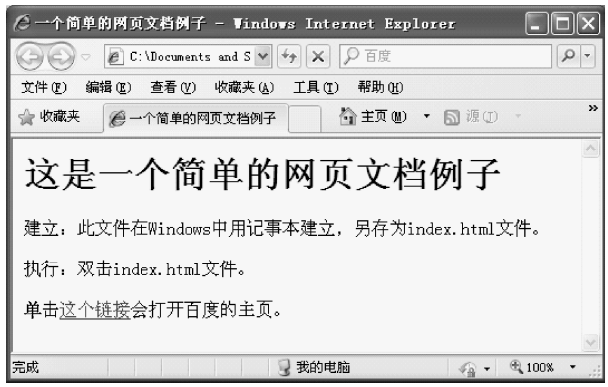


图 4-210　网页文档的显示效果

3）统一资源定位符 URL

统一资源定位符 URL（Uniform Resource Locator）是万维网中定位资源地址的方法。URL 的思想是为了使所有的信息资源位置都能用统一的方法进行描述，从而将分散的孤立信息点连接起来，实现资源的统一寻址。这里的“资源”是指互联网中可以被访问的任何对象，包括文件、文件目录、文档、图像、声音和视频等。URL 大致由三部分组成：协议、主机名、端口和文件路径。其中对于常用服务端口可以省略。其格式为

```
<协议>://<主机>:<端口>/<路径>
```

其中，<主机>部分使用域名或 IP 地址均可；<端口>如果不指定表示使用熟知端口。例如，西安交通大学主页的 URL 表示为 http://www.xjtu.edu.cn/index.html。

4）超文本传输协议 HTTP

HTTP 协议是浏览器和 Web 服务器之间用于传输网页文件的应用层协议，它的工作要依赖于传输层的 TCP 协议。HTTP 是一个面向对象的协议，当一个网页由多个对象构成（如网页中还包含一些图片、视频等对象）时，HTTP 会按每次一个对象进行传输，而每次传输都要建立一次 TCP 连接。这种工作方式不仅保证了正确传输网页文件，还能确定每次传输网页中的哪一部分，以及哪部分内容首先显示（如文本先于图形）等。

HTTP 协议定义了浏览器如何向 Web 服务器发出请求，以及 Web 服务器如何将 Web

页面返回给浏览器。当用户请求一个 Web 页面时，浏览器发送一个 HTTP 请求报文给 Web 服务器，该 HTTP 请求消息包含了所要求的网页信息。Web 服务器收到请求后，将请求的网页包含在一个 HTTP 响应报文中，并向浏览器返回该响应报文。HTTP 请求和响应的过程可描述如下（图 4-22）：

- 浏览器分析 URL。
- 浏览器向 DNS 请求解析 Web 服务器域名（如 www.xjtu.edu.cn）的 IP 地址。
- 在得到主机的 IP 地址后，浏览器与 Web 服务器建立 TCP 连接，默认端口号为 80；
- 浏览器通过 TCP 连接向 Web 服务器发送 HTTP 请求报文，该请求报文中包含了 URL 中网页文档的文件路径名部分（如 /index.html）。
- Web 服务器收到请求消息后，从本地读取网页文档并且将其封装到一个 HTTP 响应报文中，然后将 HTTP 响应报文通过 TCP 连接发送给浏览器。
- 浏览器接收到响应报文后，释放 TCP 连接。
- 浏览器从响应报文中解析出网页文档内容，对其进行解释，然后按规定的格式将网页显示在屏幕上。

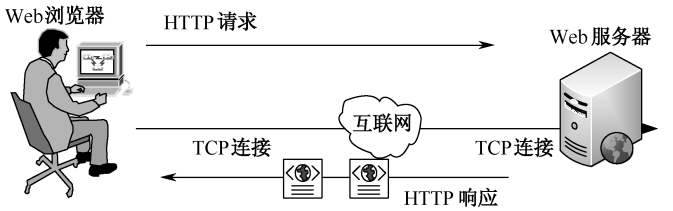


图 4-22　万维网按 B/S 方式传输网页文件

如前所述，由于一个网页往往包含有多个 HTTP 对象，所以，一次会话会重复执行上述过程中的后四个步骤很多次才能在屏幕上显示完整的页面。

3．文件传输

文件传输是互联网上与电子邮件和万维网并驾齐驱的三大应用之一。FTP 这个词即表示文件传输服务，也表示文件传输服务所使用的 FTP（File Transfer Protocol）协议。

与大多数互联网服务一样，FTP 也是一个 C/S 方式的应用系统。它允许用户从互联网上的 FTP 服务器下载（Download）各种文件到本地主机，也允许用户将本地主机上的文件上传（Upload）到远程 FTP 服务器上，达到数据备份和资源共享的目的。

FTP 客户端程序既可以使用 Web 浏览器，也可以使用专用的 FTP 程序，如 Windows 中的 FTP 程序和第三方的 CuteFTP、FlashFXP、LeapFTP 等。

FTP 服务器包括匿名 FTP 服务器和非匿名 FTP 服务器两类。匿名 FTP 服务器是任何用户都可以自由访问的 FTP 服务器，当用户登录时，使用“anonymous”作为用户名即可登录访问。对于非匿名 FTP 服务器，用户必须首先在 FTP 服务器上注册，使用合法的用户名和口令才能登录和访问。

FTP 协议基于传输层的 TCP 连接，FTP 协议运行在 20 和 21 两个端口（图 4-23）。端口 20 用于在客户端和服务器之间传输文件数据，而端口 21 用于传送文件传输过程中的各种控制信息。

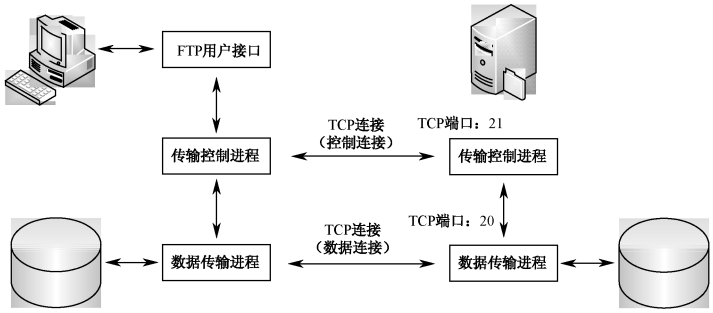


图 4-23　FTP 工作原理

FTP 进行文件传输的过程如下：

- 首先，FTP 客户程序与 FTP 服务器在 21 号端口建立 TCP 控制连接。
- FTP 客户程序向 FTP 服务器发出获取文件请求。
- 客户端和服务器各自建立一个数据传输进程。
- 客户端与服务器端的数据传输进程在 20 号接口建立 TCP 数据连接；若数据连接是由服务器方发起，则称 FTP 操作为主动模式，若数据连接是由客户端发起，则称 FTP 操作为被动模式。
- 接收并执行客户程序的命令，在 TCP 数据连接上进行文件的传输。
- 关闭 TCP 数据连接。
- 关闭 TCP 控制连接。

4．即时通信

即时通信（Instant Messaging，IM）是指能够即时发送和接收互联网消息的业务。利用即时通信工具，用户之间可以实现文字、语音、视频的实时互通交流。现在，即时通信不仅可以提供聊天功能，还集成了电子邮件、文件传输、博客、音乐、视频、游戏和搜索等多种功能。常见的即时通信软件包括 QQ、飞信、Skype、微信等。

即时通信系统最初只是一个 C/S 结构的互联网应用，但现在它已经将 P2P 和 C/S 进行结合。大多数情况下，用户首先以 C/S 方式从即时通信服务器上获取好友列表，然后用户与好友之间采用 P2P 方式进行通信。当好友之间无法建立 P2P 连接时，则好友之间采用服务器中转的方式进行通信，如图 4-24 所示。在这种系统中，即时通信服务器的主要功能是向用户提供好友目录服务，因此，使用即时通信服务的用户首先要在即时通信服务器上注册，这样通信各方才能相互定位。

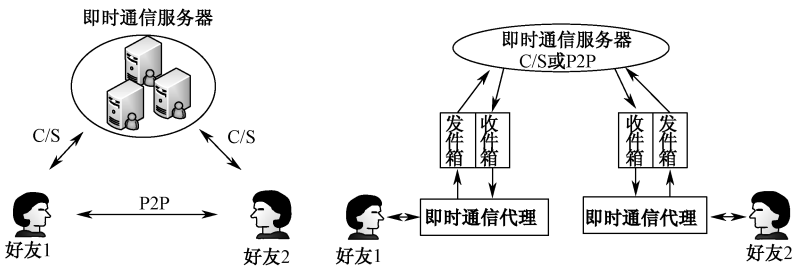


图 4-24　典型的即时通信系统结构和系统模型

即时通信系统虽然也有收件箱/发件箱（有点类似于电子邮件系统），但与电子邮件不同的是，即时消息足够短小，便于快速投递到收件箱。另外，考虑到在线人数非常多，即时通信服务内部往往是由多服务器组成的集群。

即时通信使用的传输层协议为 TCP 协议和 UDP 协议。一般情况下，注册和好友目录服务使用 TCP 协议，而好友之间的通信采用 UDP 协议。例如，QQ 在在默认状态下优先采用了 UDP 协议进行通信，其原因是 UDP 协议适用于无须应答、注重时效的网络应用，这个特点正好与 QQ 追求的目标相符。

4.3 网络安全

4.3.1 网络安全的基本概念

1. 信息系统安全

在信息时代，信息、计算机和网络是不可分割的整体。因此，信息系统安全包括了如下信息安全、计算机安全和网络安全三个方面。

- **信息安全**。指信息内容的安全。保护信息的真实性、保密性和完整性，避免攻击者利用系统的安全漏洞进行窃听、诈骗等危害合法用户利益的行为。涉及信息基础设施（各种通信设备、信道、终端和软件等）、信息资源和信息管理。
- **计算机安全**。指“为数据处理系统建立和采取的技术和管理的安全保护，保护计算机硬件、软件和数据不因偶然和恶意的原因而遭到破坏、更改和泄密”。涉及物理安全和逻辑安全两个方面的内容。物理安全指计算机系统设备及相关设备的安全；逻辑安全则指保障计算机信息系统的安全，即保障计算机中信息的完整性、保密性和可用性。
- **网络安全**。指网络上的信息安全，主要指网络系统的硬件、软件及其系统中的数据受到保护，不受偶然的或者恶意的原因而遭到破坏、更改、泄露，系统连续可靠正常地运行，网络服务不中断。

虽然不同组织和结构对信息系统安全的要求有差异，但总的目标是一致的，主要包括保密性、完整性、可用性等。

- **保密性**（Confidentiality）。指信息在存储、使用和传输过程中不泄露给非授权用户、实体或过程，或供其利用的特征。
- **完整性**（Integrity）。指数据未经授权不得随意篡改。
- **可用性**（Availability）。指系统始终处于可提供服务的状态，授权用户可以随时获得权限范围内的服务。
- **可控性**（Controllable）。指对信息的内容及传播具有掌控能力。
- **可鉴别性**（Identifiability）。指信息的真实性和用户的身份是可以鉴别的。
- **不可否认性**（Non-repudiation）。指在网络环境中，信息交换的双方不能否认其在交换过程中发送信息或接收信息的行为。

为了达到信息安全的目标，各种信息安全技术的使用必须遵守以下基本的原则：

- **最小化原则**。受保护的敏感信息只能在一定范围内被共享。并仅授予访问者必要的权限。

- **分权制衡原则**。在信息系统中，对所有权限应该进行适当地划分，使每个被授权的人只能拥有其中部分权限，使它们之间相互制约、相互监督，共同保证信息系统的安全。如果一个授权主体分配的权限过大，无人监督和制约，就可能造成“滥用权力”、“一言九鼎”的安全隐患。
- **安全隔离原则**。将信息本体与使用者分离，按照一定的安全策略，在可控和安全的前提下实施使用者对信息本体的访问。

根据以上目标和原则，为保证信息系统的安全应采取以下几个方面的具体措施：

- **访问控制**。在网络边界部署访问控制设备，启用访问控制功能；根据会话状态信息为网络行为提供明确的允许/拒绝访问的能力；对进出网络的信息内容进行过滤；在会话处于非活跃时间或会话结束后终止网络连接；限制网络最大流量数及网络连接数；采取技术手段防止地址欺骗；按访问规则，决定允许或拒绝用户对受控系统进行访问。
- **安全审计**。对网络系统中的网络设备运行状况、网络流量、用户行为等进行日志记录，日志记录包括时间、用户、事件类型、事件是否成功及其他相关的信息；对日志记录进行分析，生成审计报表；对日志记录进行保护，避免受到未预期的删除、修改或覆盖等。
- **边界完整性检查**。对非授权设备连网行为进行定位，并对其进行有效阻断；对用户非授权访问外网的行为进行定位，并对其进行有效阻断。
- **入侵防范**。在网络边界处监视对网络的攻击行为，如端口扫描、蛮力攻击、木马后门攻击、DoS 攻击、缓冲区溢出攻击、IP 碎片攻击和网络蠕虫攻击等；记录攻击源的特征信息（IP、端口、攻击类型、攻击目的、攻击时间等），在发生严重入侵事件时应提供报警。
- **恶意代码防范**。在网络边界处对恶意代码进行检测和清除；经常升级恶意代码库，更新检测系统。
- **网络设备防护**。对登录网络设备的用户进行身份鉴别，对登录者的地址进行限制；用户身份鉴别信息应具有不易被冒用的特点，口令应有复杂度要求并定期更换；限制登录失败次数，对登录失败进行处理；对网络设备进行远程管理时，采取必要措施防止鉴别信息被窃听。

2．影响信息系统安全的因素

信息系统由硬件设备、系统软件、数据资源、服务功能和用户等基本元素组成，与之相关的安全风险因素包括自然灾害威胁、系统故障、操作失误和人为蓄意破坏。而这些不安全因素则是由信息系统本身的脆弱性所决定的：

- 网络的开放性使得网络系统的协议、核心模块和实现技术是公开的，其中的设计缺陷很可能被别有用心的人所利用；网络的全球化可以使攻击者实施对网络的远程攻击；基于网络的各成员之间的信任关系可能被假冒。
- 由于网络的开放性和网络技术的普及，互联网上存在大量公开的黑客站点，获得黑客工具、掌握黑客技术越来越容易，从而导致信息系统所面临的威胁日益严重。

对信息系统的攻击主要可分为以下 5 种类型。

（1）被动攻击。攻击者非法获取敏感信息，但不对信息做任何修改。例如，监听未受保护的通信、流量分析、截获认证信息等。这种攻击方式一般不会干扰信息在网络中的正

常传输，因而也不容易被检测出来。被动攻击常用的手段如下：

- **搭线监听**。将电缆搭接在无人值守的网络传输线路上进行监听。
- **无线截获**。通过高灵敏度的接收装置接收网络辐射的电磁波，再通过对电磁信号的分析恢复原始数据。
- **其他截获**。在网络设备或主机中预置恶意程序或释放病毒程序，这些程序会将敏感信息通过某种方式泄露到外部。

（2）主动攻击。对数据进行修改或制造虚假数据。主动攻击方式如下：

- **中断**。破坏系统资源或使其不可用，造成系统因资源短缺而中断。
- **假冒**。以虚假身份获取合法用户的权限，进行非法的未授权操作。
- **重放**。攻击者对截获的合法数据进行复制，并以非法目的重新发送。
- **篡改**。将合法消息进行篡改、部分删除，或使消息延迟或改变顺序。
- **拒绝服务**（Denial of Service，DoS）。使系统拒绝合法用户对资源的访问和使用。
- **对静态数据的攻击**。包括通过穷举方式进行口令猜测、IP 地址欺骗、指定非法路由以逃避安全检测，将信息发送到指定目的站点。

（3）物理临近攻击。非法接近网络设施实施攻击活动。

（4）内部人员攻击。包括恶意攻击和非恶意攻击。恶意攻击是指内部人员有计划地窃听、偷窃或损坏信息，或干扰其他授权用户的正常访问。非恶意攻击是由于粗心、工作失职或无意间的误操作，对系统产生的破坏行为。

（5）软硬件装配攻击。采用非法手段在软、硬件的生产过程中将一些“病毒”或“木马”植入到系统中，以便日后待机攻击，进行破坏。

*4.3.2 信息安全技术

保障信息系统安全的方法很多，涉及的信息安全技术包括访问控制、数据加密、身份验证、数字签名、数字证书和防火墙等。

1．访问控制

为保障信息系统的安全，限制对信息系统的访问和接触是重要措施。信息系统的安全也可采用安全机制和访问控制技术来保障。

1）建立制定安全管理制度和措施

从管理角度来加强安全防犯。通过建立、健全安全管理制度和防范措施，约束对网络信息系统的访问者。例如，规定重要网络设备使用的审批、登记制度，网上言论的道德、行为规范，违规、违法的处罚条例等。规章、制度虽然不能防止导致数据丢失或者操作失误，但可以避免、减少一些非法行为。

2）限制对网络设施的物理接触

防止人为破坏的最好办法是限制对网络设施的物理接触。但是物理限制并不能制止窃取信息。也不能防止意外事件。

3）限制对信息的在线访问

如何限制非授权用户对信息的访问从而防止信息被盗窃或者篡改？这需要解决如何辨认是否为合法用户，尤其是从远程站点进行登录访问的用户。

限制网络访问最简单的方法就是使用用户名和口令进行认证。而这种方法的安全性取决于口令的秘密性和破译口令的难度。表 4-3 给出了口令的长度及组合对攻击难度的影响。显然，选择适当的组合方式及长度，就能使黑客破译口令的成功率大大降低。

表 4-3　口令长度及组合方式对攻击成功率的影响

口令组合策略	举例	破译需要的尝试次数	破译需要的平均时间
任何长度的姓名	Ed・Christine	2000（一个姓氏字典）	5 小时
任何长度的单词	It，electrocardiagram	6×103	7 天
两个单词的组合	Whiteknight	3.6×109	1140 年
数字字母的任意组合	JP2c2tP307k	3.7×1015	1.2×10^9 年

4）设置用户权限

如果攻击者突破了系统的安全屏蔽怎么办？通过在系统中设置用户权限可以减小非法进入系统造成的破坏。用户权限是指规定用户对文件和目录能执行哪些操作。当用户在信息系统中注册时，系统管理员会根据该用户的实际需要和角色分配给一定的权限，允许其访问指定的目录及文件，以及对这些目录和文件所能执行的操作，如只能读而不能修改和删除。用户权限是信息系统中设置的第二道安全防线。

通过配置用户权限，攻击者即使窃取到了某个用户的口令，也只能行使该用户被系统授权的操作，使损害限制在一个小范围内。

2．数据加密

1）数据加密概述

数据加密是以某种特殊的算法改变原有的信息数据，使得未授权的用户即使获得了已加密的信息，但因不知解密的方法，仍然无法了解信息的内容。由于网络的开放性，黑客很容易截获网络上传输的数据，这就使加密技术成为保障网络安全的技术手段之一。在数据加密技术中，有几个术语被经常使用：

- **明文**。原始数据。
- **密文**。加密后的数据。
- **密钥**。事先规定好的用于对明文进行加密、对密文进行解密的特殊信息。
- **加密**。把明文转换为密文的过程。
- **解密**。对密文实施与加密相逆的变换，从而获得明文的过程。
- **加密算法**。加密所采用的变换方法。

如图 4-25 所示，任何一个加密系统（密码系统）都是由明文、密文、算法和密钥组成。发送方使用加密算法，用加密密钥将明文转换成密文后发送出去。接收方收到密文后，用解密密钥将密文转换为明文。在传输过程中，即使密文被攻击者窃取，得到也只是无法识别的密文，从而起到信息保密的作用。

2）数据加密技术

加密技术一般有两种类型：“对称式”加密和“非对称式”加密。

（1）对称式加密。

即加密和解密使用同一密钥。它的优点是安全性高，加密速度快。缺点是密钥的管理

困难。在网络上很难做到绝对安全地传输密钥，也无法自动检测密钥泄漏的问题。

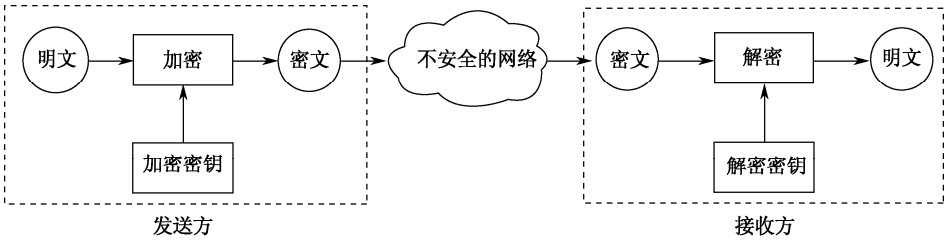


图 4-25　加密和解密过程示意图

传统的对称式加密方法可以分成两类：替代密码和换位密码。

① 替代密码。

在替代密码中，用一组密文字母来代替一组明文字母以隐藏明文，但保持明文字母的位置不变。最古老的替代密码是凯撒密码，它用 D 表示 a，用 E 表示 b，用 F 表示 c，……，用 C 表示 z，也就是说密文字母相对明文字母左移了 3 位。这里为清楚起见，一律用小写表示明文，用大写表示密文，这样明文的“cipher”就变成了密文的“FLSKHU”。更一般地，可以让密文字母相对明文字母左移 k 位，这样 k 就成了加密和解密的密钥。这种密码是很容易破译的，因为英文字母只有 26 个，所以最多只需尝试 25 次（k=1～25）即可破译密码。

较为复杂一点的密码是使明文字母和密文字母之间的映射关系没有规律可循，例如，将 26 个英文字母随意映射到其他字母上，这种方法称为单字母表替换，其密钥是对应于整个字母表的 26 个字母串。虽然初看起来这个系统是很安全的，因为若要试遍所有 26！种可能的密钥，即使计算机每微秒试一个密钥，也需要 1013 年。但事实上完全不需要这么做，破译者只要截获一部分密文，利用英文的统计特征，就很容易破译。破译的关键在于找出各种字母或字母组合出现的频率。经统计发现，英文中字母 e 出现的频率最高，其次是 t、o、a、n、i 等，最常见的两字母组合依次为 th、in、er、re 和 an，最常见的三字母组合依次为 the、ing、and 和 ion。因此，破译者首先可将密文中出现频率最高的字母定为 e，频率次高的字母定为 t，……；然后猜测最常见的两字母组、三字母组，如密文中经常出现 tXe，就可以推测 X 很可能就是 h，如经常出现 thYt，则 Y 很可能就是 a 等。采用这种合理的推测，破译者就可以逐字逐句组织出一个试验性的明文。

为了去除密文中字母出现的频率特征，可以使用多张密码字母表，对明文中不同位置上的字母用不同的密码字母表来加密。例如，任意选择 26 张不同的单字母密码表，相互间排定一个顺序，然后选择一个简短易记的单词或短语作为密钥，在加密一条明文时，将密钥重复写在明文的上面，则每个明文字母上的密钥字母即指出该明文字母用哪一张单字母密码表来加密。

例如，要加密明文“please execute thelatestscheme”，密钥为“computer”，则将“computer”重复写在报文上面，如图 4-26 所示。

c	o	m	p	u	t	e	r	c	o	m	p	u	t	…	e	r	c	o	m	p
p	l	e	a	s	e	e	x	e	c	u	t	e	t	…	s	c	h	e	m	e

图 4-26　把一段明文用密钥“computer”进行加密

于是第 1 个明文字母 p 用第 3 张（假设 a～z 分别表示顺序 1～26）单字母密码表加密，第 2 个明文字母 l 用第 12 张单字母密码表加密，……。显然，同一个明文字母因位置不同而在密文中可能用不同的字母来表示，从而消除了各种字母出现的频率特征。

② 换位密码。

换位有时又称为排列，它不对明文字母进行变换，只是将明文字母的次序进行重新排列。图 4-27 是一个用换位密码加密的例子，它的密钥必须是一个不含重复字母的单词或短语。加密时将明文按密钥长度截成若干行排在密钥下面，按照密钥字母在英文字母表中的先后顺序给各列进行编号（例如，密钥中的字母从小到大排列为 CEMOPRTU，从前到后编号依次为 1～7），然后按照列的编号按列输出明文即成密文。

C	O	M	P	U	T	E	R	密钥：COMPUTER
1	4	3	5	8	7	2	6	明文：
p	l	e	a	s	e	e	x	pleaseexecutethelatestscheme
e	c	u	t	e	t	h	e	密文：
l	a	t	e	s	t	s	c	PELHEHSCEUTMLCAE
h	e	m	e	a	b	c	d	ATEEXECDETTBSESA

图 4-27　一个用换位密码加密的例子

现代密码学也使用替代密码和换位密码的思想，但和传统密码学的侧重点不同，传统密码学的加密算法比较简单，主要通过加长密钥长度来提高加密强度，而现代密码学正好相反，它使用极为复杂的加密算法，即使破译者能够对任意数量的选择明文进行加密，也无法找出破译密文的方法。目前，使用的对称密钥密码体制是 1977 年美国国家标准局颁布的 DES（Data Encryption Standard，数据加密标准）。它采用了复杂的 DES 分组密码算法。其密钥长度为 64 位（实际只有 56 位）。其变体 TripleDES 使用了 168 位的密钥对数据进行三次加密，能够进一步提高 DES 的加密强度。

（2）非对称式加密法.

又称为公钥密码加密法。它的加密密钥和解密密钥是不同的，一个称为“公开密钥”，是公开的，可以向外界公布；另一个称为“私有密钥”，是保密的，只属于解密方持有。另外，加解密算法也都是公开的。两个密钥必须配对使用。

在网络上传输数据时，发送者用公开密钥将数据加密，接收者则使用自己的私有密钥对收到的数据进行解密。这种加密方法能很好地解决密钥分发的安全性问题。

具有代表性的公钥密码体制是 1978 年由美国人 R・Rivest、A・Shamir 和 L・Adleman 三人提出，并由他们的名字命名的 RSA 加密算法。 RSA 算法基于一个十分简单的数论事实：将两个大素数相乘十分容易，但要对其乘积进行因式分解却极其困难，因此，可以将乘积向外界公布作为加密密钥（公开密钥）。为提高加密强度，RSA 算法的密钥至少为 500 位，一般推荐使用 1024 位。但这也带来了加解密速度慢的问题。RSA 是第一个能同时用于数据加密和数字签名的算法，被认为是目前最优秀的公钥方案之一。

在实际应用中，网络传输的数据加密通常采用对称密钥和公钥密码体制相结合的混合加密体制；数据的加解密采用对称密钥体制，密钥传递则采用公钥密码体制，这样既解决了密钥管理的困难，又解决了公钥体制加解密速度慢的问题。

3）加密技术的应用

加密技术的应用领域很多，如各种系统中的用户身份认证、金融信用核实、电子商务等，都必须借助于数据加密技术，以提供相应的安全保障。

3．数字签名

数字签名是以电子形式存在于数据信息之中的数字化“签名”，可用于辨别签署人的身份，并表明签署人对数据信息中包含的信息的认可。

数字签名模拟现实生活中的笔迹签名，能够解决如何有效的防止通信双方的欺骗和抵赖行为。与加密不同，数字签名的目的是为了保证信息的完整性和真实性。为使数字签名能代替传统的签名，必须保证能够实现以下功能：

（1）接收者能够核实发送者对消息的签名。

（2）签名具有不可否认性。

（3）接收者无法伪造对消息的签名。

假设 A 和 B 分别代表一个股民和他的股票经纪人。A 委托 B 代为炒股，并指令当他所持的股票达到某个价位时，立即全部抛出。B 首先必须认证该指令确实是由 A 发出的，而不是其他什么人在伪造指令，这就需要第 1 个功能。假定股票刚一卖出，股价立即猛升，A 后悔不已。如果 A 是不诚实的，他可能会控告 B，宣称他从未发出过任何卖出股票的指令。这时 B 可以拿出有 A 自己签名的委托书作为最有力的证据，这又需要第 2 个功能。另一种可能是 B 玩忽职守，当股票价位合适时没有立即抛出，不料此后股价一路下跌，客户损失惨重。为了推卸责任，B 可能试图修改委托书中关于股票临界价位为某一个实际上不可能达到的值。为了保障客户的权益，需要第 3 个功能。

数字签名总是和报文摘要结合起来使用的，如图 4-28 所示。发送报文时，发送方用一个哈希函数从报文中生成报文摘要，然后用自己的私人密钥对这个摘要进行加密，这个加密后的摘要将作为报文的数字签名和报文及公开密钥一起发送给接收方，接收方首先用与发送方一样的哈希函数从接收到的原始报文中计算出报文摘要，接着再用接收到的公开密钥来对报文附加的数字签名进行解密，如果这两个摘要相同，那么接收方就能确认该数字签名是发送方的。

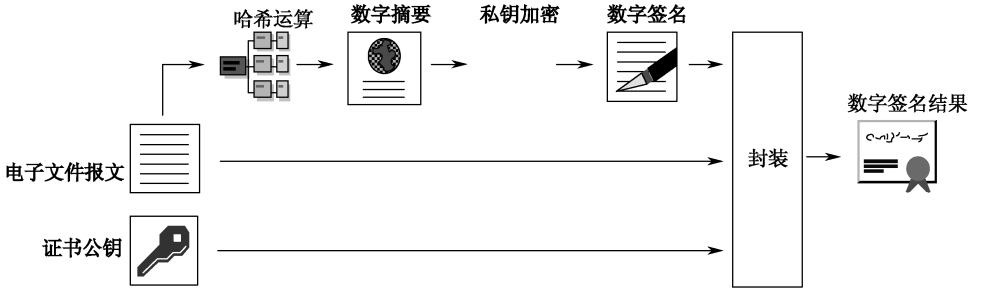


图 4-28　数字签名生成的过程

4．数字摘要

数字签名因为要用到公开密钥加密技术，因此，处理短消息比较有效，而相对于长消息则很耗时间，无法做到实时性。比较合理的做法是在数字签名前对消息先进行数字摘要，摘取其中关键的部分，使长消息变成固定长度的短消息。这个将长消息变换成短消息的运

算过程称为哈希（Hash）函数，其自变量是原始消息，函数值是变换后的短消息。因此，数字摘要就是 Hash 函数。

数字摘要的标准是 MD5（MD Standards for Message Digest）。MD5 采用单向 Hash 函数将报文“摘要”成一串 128 位的密文，这一串密文又称为数字指纹，即不同的明文摘要后的结果总是不同的，同样的明文摘要后必定一致。这样，数字摘要便成为验证明文是否是“真身”的“指纹”了。

5．数字证书

在网络上进行通信或进行电子商务活动时，使用数字证书可以防止信息被第三方窃取，也能在交易出现争执时防止抵赖的情况发生。

数字证书是指为保证公开密钥持有者的合法性，由认证机构 CA（Certification Authority）为公开密钥签发一个公开密钥证书，该公开密钥证明书称为数字证书。

数字证书是互联网通信中标志通信各方身份信息的一系列数据，是一个经证书授权中心数字签名的包含公开密钥拥有者信息及公开密钥的文件。最简单的证书包含一个公开密钥、名称及证书授权中心的数字签名。

在图 4-27 中，若电子文件报文是包含公开密钥持有者信息的文件，那么数字签名的结果就是数字证书。只不过，用于加密数字摘要的私有密钥是由 CA 所拥有。

数字证书提供了一种在互联网上验证身份的方式，其作用类似于司机的驾驶执照或日常生活中的身份证。它是进行安全通信的必备工具，它保证信息传输的保密性、数据完整性、不可抵赖性，以及交易者身份的确定性。

数字证书由权威、公正的认证机构 CA 颁发和管理。在国际电信联盟 ITU 制定的 X.509 标准中，规定了数字证书主要包含以下内容：

- 证书所有人的名称。
- 证书所有人的公开密钥。
- 证书发行者对证书的签名。
- 证书的序列号，每个证书都有一个唯一的证书序列号。
- 证书公开密钥的有效日期等。

使用网上银行进行各种金融交易时，为保证交易的安全，往往需要一个称为 U 盾的 USB 设备，它外形类似于一个 U 盘，在网上银行操作时需要插在计算机的 USB 接口上。这个 U 盾其实就是数字证书。

*4.3.3　网络安全防护

几乎所有上网用户都经受过计算机病毒和木马的袭扰，对于一些机构的计算机网络，还会受到入侵攻击。本节将对这些方面进行简介，并使读者了解如何检测、避免或减轻这些隐患的威胁。

1．计算机病毒和木马

计算机病毒（Computer Virus）是一种人为编制出来嵌入到其他软件中的程序段。它具有感染性、隐蔽性、潜伏性、破坏性、可触发性、攻击的主动性、不可预见性等特征。

木马（Trojan）也属于一种计算机病毒。这种“木马”是借用于古希腊传说中的著名计策木马计。木马程序与一般的病毒不同，它不会自我繁殖，也并不“刻意”地去感染其他文件，它是通过将自身伪装成一个非常吸引人们眼球的应用（如一个好玩的小游戏，一段视频，一个非常稀罕的资源等），吸引用户下载或执行，这时木马就会以插件形式嵌入到用户主机的操作系统、浏览器等软件中，在受害者电脑中打开了一个后门，使施种者可以任意毁坏、窃取被害者的敏感信息（如银行账号、密码等），甚至远程操控被害者的计算机去做危害网络的操作。

对计算机病毒的预防可以通过严格的管理措施和技术手段。技术手段包括安装设置防火墙、安装实时监测的杀病毒软件、不要轻易打开陌生人传来的页面链接等。

2．入侵检测

入侵检测（Intrusion Detection）是对入侵行为的检测。它通过收集和分析网络行为、安全日志、审计数据、其他网络上可以获得的信息，以及计算机系统中若干关键点的信息，检查网络或系统中是否存在违反安全策略的行为和被攻击的迹象。

入侵检测作为一种积极主动地安全防护技术，提供了对内部攻击、外部攻击和误操作的实时检测和保护，在网络系统受到危害之前进行拦截和响应，包括切断网络连接、记录事件和报警等。因此，入侵检测系统（Intruder Detect System，IDS）被认为是防火墙之后的第二道安全门。

入侵检测通过执行以下任务来实现：

- 监视、分析用户及系统活动。
- 识别反映已知进攻的活动模式并向相关人士报警。
- 异常行为模式的统计分析。
- 评估重要系统和数据文件的完整性。
- 操作系统的审计跟踪管理，并识别用户违反安全策略的行为。

入侵检测系统有以下几种类型：

1）基于主机的入侵检测系统

主要使用主机操作系统的审计、跟踪日志作为数据源，某些入侵检测系统也会主动与主机系统进行交互以获得不存在于系统日志中的信息以检测入侵。这种类型的检测系统不需要额外的硬件，对网络流量不敏感，效率高，能准确定位入侵位置并及时进行反应，但是占用主机资源，依赖于主机的可靠性，所能检测的攻击类型受限。不能检测网络攻击。

2）基于网络的入侵检测系统

通过被动地监听网络上传输的原始流量，对获取的网络数据进行处理，从中提取有用的信息，再通过与已知攻击特征相匹配或与正常网络行为原型相比较来识别攻击事件。此类检测系统不依赖操作系统作为检测资源，可应用于不同的操作系统平台；配置简单，不需要任何特殊的审计和登录机制；可检测协议攻击、特定环境的攻击等多种攻击。但它只能监视受监视网段的活动，无法得到主机系统的实时状态，精确度较差。大部分入侵检测工具都是基于网络的入侵检测系统。

3）分布式的入侵检测系统

这种入侵检测系统一般为分布式结构，由多个部件组成，在关键主机上采用主机入侵

检测，在网络关键结点上采用网络入侵检测，同时分析来自主机系统的审计日志和来自网络的数据流，判断被保护系统是否受到攻击。

一个入侵检测系统通常包括四个核心组件：事件产生器、事件分析器、响应部件和事件数据库，如图 4-29 所示。

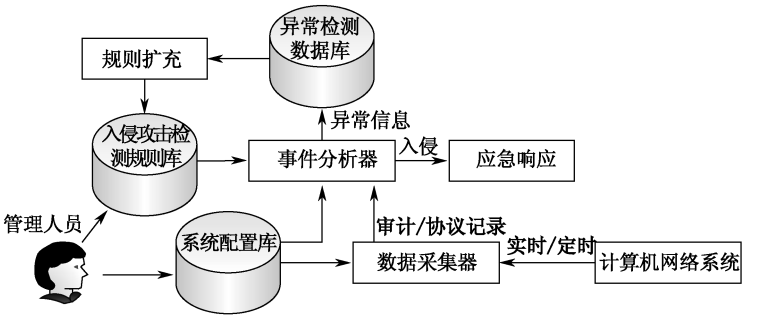


图 4-29　入侵检测系统基本结构

事件产生器的目的是从网络中获取事件，并向系统的其他部分提供此事件。事件分析器分析得到的数据，并产生分析结果。响应单元则是对分析结果做出反应的功能单元，它可以做出切断连接、改变文件属性等强烈反应，也可以只是简单的报警。事件数据库是存放各种中间和最终数据的地方，它可以是大型数据库，也可以是文本文件。

3．防火墙

防火墙是指由软件和硬件设备组合而成的，在机构网络和互联网之间建立的一个安全网关（Security Gateway），能够保护内部网免受非法用户的侵入（图 4-30）。防火墙主要由服务访问规则、验证工具、包过滤和应用网关 4 个部分组成。

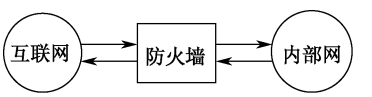


图 4-30　防火墙位于内网与互联网之间

市场的防火墙产品非常多，主要分类有“包过滤型”、“状态检测型”和“应用代理型”三大类。

1）包过滤型防火墙

包过滤是防火墙最基本的实现形式，它控制哪些数据包可以进出网络，而哪些数据包不允许进出网络。可依据以下三类条件允许或阻止数据包通过防火墙：

- 包的源地址及源端口。
- 包的目的地址及目的端口。
- 包的传送协议，如 FTP、SMTP、HTTP、rlogin 等。

例如，包过滤能让我们进行以下情况的操作：

- 不让任何用户从外部网络远程登录内网中的数据库服务器。
- 禁止 263.net 的电子邮件用户使用 SMTP 协议向内网发送电子邮件。
- 只允许外部网络用户访问内部网的 Web 服务器，而不允许访问数据库服务器。

2）状态监测型防火墙

状态检测型防火墙可以动态地设置包过滤规则的方法。状态检测型防火墙对通过其建立的每一个连接都进行跟踪，并且根据需要可动态地在过滤规则中增加或更新规则条目。

状态监视技术通过“状态监视”组件，在不影响网络安全正常工作的前提下采用抽取

相关数据的方法对网络通信的各个协议层次实行监测，并根据过滤规则做出安全决策。

状态监视技术在保留了对每个数据包的头部、协议、地址、端口、类型等信息进行分析的基础上，增加了“会话过滤”（Session Filtering）功能。在每个连接建立时，防火墙会为这个连接构造一个会话状态，里面包含了建立连接的数据包的所有信息，以后防火墙会基于这个状态信息对此连接进行检测。这种检测的高明之处是能对每个数据包的内容进行监视，一旦建立了一个会话状态，则此后的数据传输都要以此会话状态作为依据。例如，一个连接的数据包源端口是 8000，那么在以后的数据传输过程里防火墙都会审核这个包的源端口还是不是 8000，否则这个数据包就被拦截。状态监视可以对包内容进行分析，从而摆脱了传统防火墙仅局限于包头部信息检测的缺点，而且这种防火墙不必开放过多端口，进一步杜绝了可能因为开放端口过多而带来的安全隐患。

3）应用代理型防火墙

应用代理（Application Proxy）型防火墙是过滤型防火墙与应用网关（Application Gateway）配合使用，共同组成的防火墙系统。

应用代理型防火墙工作在网络体系结构的应用层。其特点是完全“阻隔”了网络通信流，通过对每种应用服务编制专门的代理服务程序，实现监视和控制应用层通信流的作用。

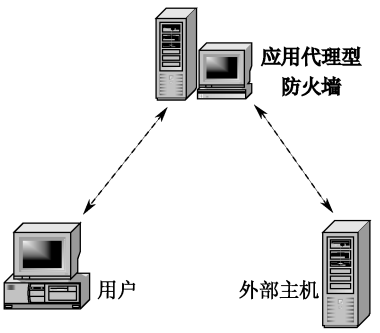


图 4-31　应用代理防火墙的实现

代理服务是指定一台有访问互联网能力的主机作为内网客户端的代理去与互联网中的主机进行通信。代理服务器判断从客户端来的请求并决定哪些请求允许传送而哪些应被拒绝。当某个请求被允许时，代理服务器就代表客户端与互联网中的主机进行通信，并将从客户端来的请求传送给互联网中的主机，或将互联网中的主机的应答传送给客户端。

代理服务器作为内部网络用户的“代言人”，这就使得内部网络完全被隐藏起来，从而进一步阻止了黑客从互联网远程攻击内网的企图。应用代理防火墙的实现如图 4-31 所示。

习题

一、填空题

1．计算机网络按照其规模大小和延伸距离远近划分为（　　）、（　　）和（　　）。

2．网上的站点通过点到点的链路与中心站点相连，具有这种拓扑结构的网络称为（　　）。

3．计算机网络中常用的有线传输介质有　（　　）。

4．网络协议的关键要素包括语法、（　　）和（　　）。

5．TCP/IP 参考模型共分为 4 层，分别是（　　）、（　　）、（　　）和（　　）。

6．互联网上的每一台主机都有一个唯一的、可识别的主机地址，称为（　　）。

7．万维网 WWW 的三个组成部分是（　　）、（　　）和（　　）。

8．数据加密技术一般有两种类型，分别是（　　）加密和（　　）加密。

二、选择题

1．以下所列出的，哪些是计算机网络的功能设备？（　　）

A．主机　　B．鼠标　　C．电话机　　D．FTP 服务器

E．双绞线　　F．USB 线　　F．通信卫星　　H．电话线

I．网络交换机　　J．路由器　　K．键盘　　L．U 盘

M．网络接口卡　　N．光纤　　O．显示器　　P．打印机

Q．多媒体音箱　　R．调制解调器　　S．智能手机　　T．平板电脑

2．以下哪两个 IP 地址属于同一子网（子网掩码为 255.255.192.0）？（　　）

A．150.20.115.133　　B．150.20.190.2

C．150.20.192.59　　D．150.20.215.133

3．DNS 系统用于以下哪项任务？（　　）

A．将 IP 地址转换为 MAC 地址　　B．将域名转换为 MAC 地址

C．将域名转换为 IP 地址　　D．将 IP 地址转换为 MAC 地址

4．下列攻击中，（　　）属于主动攻击。

A．无线截获　　B．搭线监听

C．拒绝服务　　D．流量分析

三、简答题

1．现代计算机网络为什么要将报文分割成一个个分组来进行传输？说出你认为最重要的一个理由。

2．采用全互连拓扑结构建设一个具有 500 个结点的广域网。假定网络中结点之间的平均距离为 50 公里，每公里的线路成本是 1 万元。建设此广域网的线路总成本是多少？通过这个结果能得到什么结论？

3．如果在家里构建了一个能够连接互联网的家庭局域网。请构思一个能够利用家庭局域网为在家居生活带来方便的应用。简要说明你的构思大致的实现方法。

4．哪一种网络应用模式对客户端的要求最低？哪一种网络应用模式对所有主机的要求是等同的？

5．互联网的体系结构是怎样的？画出互联网体系结构的层次图，并在图上标注每层的名字。

6．中国一家公司的经理要与德国一家公司的经理进行商务谈判。请将谈判过程的机制用层次结构表示，给每个层次用一个贴切的词汇命名，说明每个层次的功能和对等层之间的协议是什么，最后详细描述在这种机制下的商谈过程。已知条件如下：

（1）谈判策略已经由双方的董事会各自确定，由双方的经理亲自掌握。

（2）中方经理不懂德语，德方经理也不懂汉语，但双方都可以聘请翻译人员。

（3）翻译人员只负责语言翻译，不涉及商务。

（4）双方的通信手段只能使用传真，只有秘书会用传真。

7．一个具有 5 层体系结构的网络，其每一层添加的报文头部长度为 20B。假定发送方的用户要传输一份 500B 的报文给对方（不考虑报文分段），物理介质上传输的总位数是多少？网络的传输效率是多少？接收方用户收到的报文长度是多少？

8．用户主机上打开了两个 IE 浏览器窗口，浏览同一个网站的不同网页。该网站的 Web 服务器如何知道将网页发送到哪个 IE 浏览器窗口？

9．一般情况下，通过域名访问一个网站需要访问几次根域名服务器就能查找到该域名服务器的 IP 地址？

10．以太网中的主机在发送时为什么还要监听介质？

11．局域网设置网关的作用是什么？

12．在发送邮件过程中要建立几个 TCP 连接？

13．在一次 FTP 传输中要建立几个 TCP 连接，每个 TCP 连接的作用分别是什么？

14．假设密钥 k=8，用替代密码将明文“XIANJIAOTONGUNIVERSITY”加密。

15．在网络购物过程中，用到了哪些安全技术？它们分别用于网络购物的哪个步骤？

16．在网上传输的数字证书中包括了报文明文、证书公钥和数字签名。但证书公钥并没有被加密，而是直接封装在证书中。不加密的原因是什么？如果对公钥加密再封装到证书中会出现什么问题？

17．防火墙有哪些种类？哪一种防火墙的安全性更好一些？为什么？

第 5 章　Visual Basic 程序设计

引言

计算机程序是一系列的指令，它能使计算机执行特定的任务。编程语言，如 Visual Basic 2008 用于将我们能懂得指令翻译成计算机可以理解和执行的步骤。深入计算机的最底层，作为计算机“心脏”的微处理器，只能理解数字指令（机器指令）。这种处理器所能理解的指令是极简单的命令，这些命令大多数只能处理在存储地址之间移动数据。这些处理器懂得的命令称为机器语言，即 PC 可使用的最基本的语言。

机器语言称为低级语言，因为它是处理器能够懂得的最低级的方式。用机器语言编写程序是一项极烦琐的任务。幸运的是，用户不必用它来编写计算机程序。有更高级的编程语言已经开发出来，使我们能够编写程序。这些高级编程语言允许程序员以类似英文的方式编写指令，然后将指令转化成处理器能懂的含有机器语言指令的程序。

本节讲述基本的 Visual Basic 2008 编程语言。但此处不是系统讲述 Visual Basic 程序设计的书，在此仅讲述最基本的 Visual Basic 语句，也就是 Visual Basic 一个子集，用于支持后续章节的学习。如果对 Visual Basic 编程感兴趣，请参考其他有关的编程书籍。

教学目的

- 掌握 Visual Basic 程序的基本结构。
- 掌握变量、数组和表达式及运算符的基本用法。
- 掌握控制语句的用法。
- 掌握过程的用法。
- 了解面向对象的编程方法。
- 熟悉 Visual Studio 2008 编写 Visual Basic 程序的方法。

5.1　程序设计基础

5.1.1　什么是程序设计

计算机之所以比电视机、DVD、计算器等其他电子设备功能更灵活，是因为计算机软件的“可编程”，也就是说，同样的硬件配置，加载不同的软件就可以完成不同的工作。

当用户使用计算机来完成某项工作时，会面临两种情况：一种是可以借助现成的应用软件完成，如文字处理可使用 Word，表格处理使用 Excel，科学计算可选择 MATLAB，绘制图形可使用 PhotoShop 等；另一种情况是，没有完全合适的软件可供使用，这时就需要

使用计算机语言来编制完成某个特定功能的程序，这就是程序设计（Programming）。

考虑这样一个任务：统计大学中一个班学生的考试成绩，并选出优秀学生。

可以看出，这是一个很简单的任务。如果由人去做，是一件比较轻松的事，只需要几张纸和一枝笔，一个小时就完成了。对这样简单的任务，可以不需要计算机去做。但如果将任务修改为要求对一个大学中的 4 万名学生的英语成绩进行管理，并分别统计出 100 至 60 各个分段的人数，并对过去 5 年中的数据进行对比分析。这个任务由人工完成就比较麻烦了，就完全值得设计计算机程序了。

功能完善的商业程序一般都是比较大的，一个字处理软件就包含 75 万行代码，而按照美国国防部的标准，少于 10 万行代码称为小程序，超过 100 万行才是大程序。为便于理解，我们还是以微小的程序作为例子来介绍程序设计的概念。

计算机程序设计是用计算机语言编写一些代码（指令）来驱动计算机完成特定的功能，是问题求解过程的关键步骤之一。我们已经知道，对复杂的系统性问题的求解，需要经过需求分析、软件设计、程序编制和调试、系统测试、文档编写等一系列活动，甚至还包括硬件系统配置、人员配置、开发方法及开发工具选择等相关的各种任务管理。而对于小型问题（或称算法类问题），这个过程可以简化为问题描述、算法设计、代码编制及调试运行。

5.1.2 程序设计语言

在过去的几十年里，人们根据描述问题的需要而设计了数千种专用和通用的计算机语言，有的语言是为了编写系统软件而重在提高效率（如C 语言）；有些是为了提高程序设计速度，面向商业应用（如 COBOL 和 DBase）；还有些语言是为了用于教学（BASIC 和 Pascal 都属于此类）。这些语言中只有少部分得到了比较广泛的应用。

1．计算机语言的分类

对程序设计语言的分类可以从不同的角度进行，如面向机器的程序设计语言、面向对象的程序设计语言、面向过程的程序设计语言等。其中，最常见的分类方法是根据程序设计语言与计算机硬件的联系程度，可以将其分为机器语言、汇编语言和高级语言三种类型。前两种类型的语言紧密依赖于计算机硬件，有时又称为低级语言；而高级语言与计算机硬件关系较小。可以说，程序设计语言的演变经历了由低级向高级的发展过程。

1）机器语言（Machine Language）

机器语言是直接用机器指令的集合作为程序设计手段的语言，而机器指令是以计算机所能理解和执行的以“0”和“1”组成的二进制编码表示的命令，它是所有语言中唯一能够被计算机直接理解和执行的指令。如第 2 章中所述，机器指令由操作码和操作数组成，其具体的表现形式和功能与计算机系统的结构相关联。

机器语言是面向机器的语言，其优点是计算机能够直接识别、执行效率高；缺点是因与具体机器相关，可移植性很差。且由于是由 0 和 1 这样的编码表示，在记忆、书写、编程、可读性等方面都比较困难。表 5-1 分别用二进制和十六进制数表示的机器指令代码，功能是求两数之和。由此可以看出，机器指令使用起来是很困难的。

表 5-1 求两数之和的机器语言指令

二进制表示	十六进制表示
10100000000000000000000	A10000
00000011000001101100100000000000	0306C800
10100011110011000000000	A3CC00

2）汇编语言（Assembly Language）

为了克服机器语言的缺点，人们采用了助记符与符号地址来代替机器指令中的操作码与操作数，如用 ADD 表示加法操作，用 SUB 表示减法操作，且操作数可用二进制、八进制、十进制和十六进制数表示。这种表示计算机指令的语言称为汇编语言。汇编语言也是一种面向机器的语言，但计算机不能直接执行汇编语言程序。用它编写的程序必须经过汇编程序翻译成机器指令后才能在计算机上执行。目前，由于它比机器语言可理解性好，比其他语言执行效率高，许多系统软件的核心部分仍采用汇编语言编制。

表 5-1 中的机器语言指令代码若用汇编语言编写，可表示为如下 3 行指令：

```
MOV AX，DATA1
ADD AX，DATA2
MOV SUM，AX
```

3）高级语言

高级语言是更接近自然语言、更接近数学语言的程序设计语言，是面向应用的计算机语言，与具体的计算机硬件平台无关。它的主要优点是符合人类叙述问题的习惯，而且简单易学。目前的大部分程序设计语言都属高级语言，其中，使用较多的有 BASIC（Visual Basic）、Pascal（Delphi）、FORTRAN、COBOL、C、C++、Java 等。

用 Visual Basic 语言完成表 5-1 的程序功能只需要下边一行语句：

```
SUM = DATA1 + DATA2;
```

可以看出，这很接近于自然语言，便于人理解。

目前，高级语言正朝着非过程化发展，即只需告诉计算机“做什么”，“怎样做”则由计算机自动处理。高级语言的发展将以更加方便用户使用为宗旨。

2．程序语言的语法和语义

计算机语言有多种，不同的语言在语法和语义上都存在一定的差异。通俗地讲，语义是程序语言所表示功能的描述。例如，给一个数赋值、比较两个数大小、在屏幕上显示文字等。

不同的计算机语言可能都具有某些相同的功能定义，但在表现形式上却有不同，也就是语法不同。不同计算机语言的区别主要表现在在语法（词法）上，即对同一种功能（相同语义），不同的语言可能有不同的表现形式。例如，给一个变量赋值，下边的两种语言就有两种不同的表示方法：

```
sum := eng + math          Pascal 语言的赋值语句
sum = eng + math;          C++语言的赋值语句
```

又如：要表示“如果数学成绩（math）和英语成绩（eng）都大于 80 就显示姓名（name）和分数合计（sum）”。C++语言的语句如下：

```
if (math>80 && eng >80)
   cout << "第一名: "<<name<<sum;
```

而如果用 Basic 语言表示，则

```
IF math>80 AND eng >80 THEN
   PRINT name1,sum1
END IF
```

可以看出，不同的语言用不同的符号表达了完全相同的含义。但很多时候，语言的差异还体现在语义上，即功能的描述上。例如，在 C++和 Pascal 中的指针，在很多语言中就没有对等的体现。

3．程序执行的起始点

程序都会从起始点开始执行，但不同的语言对起始点的处理方法有所不同。BASIC 语言中，程序从第一条语句开始执行，不论它是什么（这应当是最直观、最容易理解的方式了!）；在 C++程序和 Visual Basic 控制台程序中，程序会从 main 函数的第一条语句开始执行，而不论 main 函数处于程序的什么位置（如果没有 main 函数，则无法运行）。

以下是一个 Visual Basic 程序编写的在屏幕上显示“Hello World!”的程序段。

```
Sub main()
{
    Console.WriteLine("Hello World")
  }
```

程序从 main 函数的第一条语句开始执行

程序从起始点开始，按照程序员书写的顺序一条条执行指令。第一条语句先执行，接下来是第二条，…，一直到最后程序结尾。如果遇到一个子程序，则中断当前程序而转去执行子程序，执行完返回刚才的断点继续执行。

除了顺序执行外，程序执行还有分支、循环等多种控制。程序的控制结构将在 5.4 节中讨论。

5.1.3 程序的编译

在计算机语言中，用除机器语言之外的其他语言书写的程序都必须经过翻译，变成机器指令，才能在计算机上执行。因此，各种用于计算机程序设计的语言，都必须配备相应的“翻译程序”。

编译是指将用高级语言编写好的程序（又称源程序、源代码），经编译程序翻译，形成可由计算机执行的机器指令程序（称为目标程序）的过程。如果使用编译型语言，必须把程序编译成可执行代码。因此，编制程序需要三步：写程序、编译程序和运行程序。一旦发现程序有错，哪怕只是一个错误，也必须修改后再重新编译，然后才能运行。幸运的是，只要编译成功一次，其目标代码便可以反复运行，并且基本上不需要编译程序的支持就可以运行。

5.2 变量及数据类型

在不同的程序设计语言中，数据类型的规定和处理方法是不同的。数据类型用来描述真实世界中不同类别的信息。Visual Basic 提供了一些预先定义好的数据类型，部分数据类

型如表 5-2 所示。

表 5-2 Visual Basic 的数据类型

数据类型	示例	存储分配	取值范围
Boolean	True	2 字节	True 或 False
Byte	122	1 字节	0～255（无符号）
Char	C	2 字节	0～65535（无符号）
Date	04/23/1972 02:00 PM	8 字节	0001 年 1 月 1 日凌晨 0:00:00 到 9999 年 12 月 31 日晚上 11:59:59。
Decimal	3.1415926 34567888	16 字节	0～+/–79 228 162 514 264 337 593 543 950 335 之间不带小数点的数；0～+/–7.9 228 162 514 264 337 593 543 950 335 之间带 28 位小数的数；最小非零数为+/–0.000 000 000 000 000 000 000 000 000 1(+/–1E–28)
Double	22.34E22	8 字节	负数取值范围为 –1.79769313486231E+308 ～ –4.940 656 458 412 47E–324；正值取值范围为 4.940 656 458 412 47E–324～1.797 693 134 862 31E+308
Integer	1234567	4 字节	–2 147 483 648～2 147 483 647
Long	1234567890	8 字节	–9 223 372 036 854 775 808～9 223 372 036 854 775 807
Object		4 字节	任何类型都可以存储在 Object 类型的变量中
Short	23456	2 字节	–32 768～32 767
Single	423E12	4 字节	负值取值范围为–3.402 823E+38～–1.401 298E–45；正值取值范围为 1.401 298E–45～3.402 823E+38
String	Hello	取决于实现平台	0 到大约 20 亿个 Unicode 字符

对表格的说明如下：

- Boolean 变量以 s16 位（2B）的数值形式存储，但取值只能是 True 或是 False。使用关键字 True 与 False 将 Boolean 变量赋值为这两个状态中的一个。在将数值数据类型转换为 Boolean 值时，0 会转换为 False，而其他所有值都将转换为 True。在将 Boolean 值转换为数值类型时，False 将转换为 0，True 将转换为–1。
- Char 变量以无符号的 16 位（2B）数字的形式存储，取值范围为 0～65 535。每个数字代表一个 Unicode 字符。
- Date 变量以 8B 整数的形式存储，表示从 1 年 1 月 1 日到 9999 年 12 月 31 日的日期从凌晨 0:00:00 到晚上 11:59:59 的时间。Date 值必须以数字符号（#）括起来，格式必须为 m/d/yyyy，如#5/31/1993#。
- 22.34E22 是科学计数法，表示为 22.34×10^{22}。
- Byte、Integer、Long 和 Short 均可存放一个整数，它们的取值范围不同，占有的空间大小也不一样。实际编程时，可根据需要选用。
- Decimal、Double 和 Single 均存放一个不同范围的实数。

变量是在程序运行过程中其值是可以变化的量。使用变量前，一般必须先声明变量名和其类型，以确定为它分配多大的存储单元。在 Visual Basic 中用以下方式来声明变量及类型。

```
Dim 变量名 As 类型
```

其中，类型可使用表 5-1 中所列出的数据类型或用户自定义的类型名。例如：

```
Dim myName As String
Dim age As Integer
```

可以用一个声明语句声明多个同变量，下面的语句声明两个整型变量 A 和 B，例如：

```
Dim A, B as Integer
```

常量是在程序运行中不变的量。声明常量的语法如下：

```
Const 常量名 [As 类型] = 表达式
```

其中，As 类型为可选项，说明了该变量的数据类型。省略该项，数据类型由表达式决定。表达式的值即为该常量的值（表达式将在 5.3 节讲述）。例如：

```
Const PI As Single= 3.14159        '声明了常量 PI，代表 3.14159，Single 型
```

除了使用 Visual Basic 内置的数据类型之外，还可以通过现有的数据类型的组合来创建自定义的数据类型。创建的方式是使用 Structure 和 End Structure 关键字。例如，通过使用两个 Single 型变量，定义新的 Point 型变量，来表示平面上一个点的坐标：

```
Structure Point
       Dim x As Single
       Dim y As Single
End Structure
```

定义了一个新的 Point 型变量后，便可以申明一个 Point 型的变量。要使用 Point 型的变量，必须单独对其包含的每一个数据（此处是 x 和 y，称为 Point 的成员变量）分别存取。格式是变量名加上句点后跟成员变量的名字。例如：

```
Dim p1 As Point
p1.x = 23.3
p1.y = 44.8
```

5.3 运算符及表达式

Visual Basic 通过运算符、变量等组合成表达式，实现编程中所需的大量操作。Visual Basic 中的运算符分为算术运算符、关系运算符和逻辑运算符等，如表 5-2、表 5-3 和表 5-4 所示。

5.3.1 赋值运算符

表达式由变量、常量、运算符和圆括号按一定的规则组成。要掌握表达式首先要理解运算符的使用。其次表达式计算出来的值通常要存放到变量中，这是通过赋值语句来完成的。

赋值语句是任何程序设计中最基本的语句。它的作用是把右边表达式的值赋给左边的变量，Visual Basic 使用赋值号“=”来赋值，其语法如下：

```
变量名=表达式
```

表达式的计算结果类型应与变量名的类型一致，即同时为数值型或同时为字符型。当数值型具有不同的精度时，强制转换成左边的精度。

特别需要注意的是，赋值运算符不是数学中的等号。其左边必须是一个变量，用来存放右边的运算结果，x+2=y+5 这样的写法是错误的。

5.3.2 算术运算符

常用的算术运算符如表 5-3 所示。

表 5-3 算术运算符（表中 A=3）

运算符	含义	优先级	例子	结果
^	乘方	1	A^2	9
–	负号	2	–A	–3
*	乘	3	A*A	9
/	除	3	10/A	3.33333333333333
\	整除	4	10\A	3
Mod	取模	5	10 Mod A	1
+	加	6	10+A	13
–	减	6	A–10	–7

例 5-1 使用 Visual Basic 语句表达下面的式子。

$$L=\sqrt{(x1-x2)^2+(y1-y2)^2}$$

分析：先声明变量，假定为 Single；再写出表达式。

程序：

```
Dim x1 As Single = 1.5
Dim x2 As Single = 2.1
Dim L As Single
Dim y1 As Single = 2.9
Dim y2 As Single = 4.7
L = ((x1 - x2) ^ 2 + (y1 - y2) ^ 2) ^ (1 / 2)
```

在进行字符串运算时，使用的字符串运算符有两个：“＆”、“+”，运算结果都是将两个字符串拼接起来。在字符串变量后使用运算符“＆”时，变量与运算符“＆”间应有一个空格。例如：

```
"高级"+"编程"          '结果为"高级编程"
"This is a" & "VB.NET"  '结果为"This is a VB.NET"
```

在使用时，连接符“＆”与“+”的区别如下：

- “+”。连接符前后的表达式应均为字符串，若均为数值则进行算术加运算。若一个为字符串，另一个为数值则会出错。
- “＆”。连接符前后的表达式不管是字符串还是数值，进行连接操作前，系统先将表达式转换成字符串，然后再连接。

例如：

```
"aabbcc"+123456        '出错
"aabbcc" & 123456      '结果为"aabbcc123456"
```

5.3.3 关系运算符

关系运算符是双目运算符，作用是将两个表达式进行比较，若关系成立，则返回 True，否则返回 False。表达式可以是数值型、字符型。常用的关系运算符如表 5-4 所示。

使用关系运算符时应注意以下规则：

- 如果两个表达式是数值，则按其大小比较。
- 如果两个表达式是字符或字符串，则按字符的 ASCII 码值从左到右一一比较，即首先比较两个字符串的第 1 个字符，其 ASCII 码值大的字符串大，如果第 1 个字符相同，则比较第 2 个字符，以此类推，直到出现不同的字符为止。

- 关系运算符的优先级相同。

表 5-4　关系运算符

运算符	含义	例子	结果
=	等于	"ABCDEF"="ABS"	False
>	大于	"ABCDEF">"ABS"	False
>=	大于等于	"bc">="abcdef"	True
<	小于	23<3	False
<=	小于等于	23<=3	False
<>	不等于	77<>99	True
Like	字符串匹配	"ABCDEFG" Like "*DE*"	true

5.3.4　逻辑运算符

逻辑运算符除 Not 是单目运算符外，其余都是双目运算符，作用是将表达式进行逻辑运算，结果是逻辑值 True 或 False，如表 5-5 所示。

表 5-5　逻辑运算符

运算符	说明	优先级	说明	例子	结果
Not	取反	1	当表达式为 False 时，结果为 True	Not F	Ture
And	与	2	两个表达式均为 True 时，结果才为 True	T And F T And T	False Ture
Or	或	3	两个表达式中有一个为 True 时，结果为 True	T Or F F Or F	Ture False

例 5-2　变量 Y 为一整数，表示年代。判断 Y 是否是闰年，如是则表达式为 True，否则为 False。写出相关的判断语句。

分析：依照题意，闰年的标准是能被 4 整除且不能被 100 整除的为闰年。能被 400 整除的是闰年。

程序：

```
Dim Y As Integer = 2011
Dim leapYear As Boolean
leapYear = ((Y Mod 4 = 0) And (Y Mod 100 <> 0)) Or (Y Mod 400 = 0)
```

5.3.5　表达式

表达式由变量、常量、运算符和圆括号按一定的规则组成。表达式通过运算后有一个结果，运算结果的类型由数据和运算符共同决定。表达式的书写规则如下：

- 乘号用“*”表示，并且不能省略。例如，a 乘以 b 应写成 a*b。
- 括号必须成对出现，均使用圆括号。
- 表达式从左到右在同一基准上书写，无高低、大小之分。

例如：已知数学表达式 $\frac{\sqrt{(3x+y)-z}}{(xy)^3}$，写成 Visual Basic 表达式为

((3*x+y)–z)^(1/2)/(x*y)^3

在算术运算中，如果表达式具有不同的数据精度，则 Visual Basic 规定运算结果的数据类型采用精度高的数据类型，即

```
Integer < Long < Single < Double
```

但当 Long 型数据与 Single 型数据运算时，结果为 Double 型数据。

关系运算符的优先级相同。当一个表达式中出现了多种不同类型的运算符时，不同类型的运算符优先级如下：

```
算术运算符 > 关系运算符 > 逻辑运算符
```

实际上，对于多种运算符并存的表达式，可增加圆括号，改变优先级可使表达式的层次更清晰。

5.4 控制语句

对一个程序员来讲，程序设计工作的一个主要内容就是如何将设计好的算法用某种程序语言来描述。换句话说，就是如何组织程序的结构。在第 3 章关于计算机工作原理的介绍中曾经描述过，程序是指令序列的集合。按照计算机“本能”的工作过程，当执行开始、取出第一条指令后，程序计数器 PC 会加 1，继续读取下一条指令，并依次下去。这样的执行方式称为顺序执行方式，相应的程序结构称为顺序结构。

程序的执行除了这种顺序方式外，还有分支、循环等多种执行方式。程序模块，可以是一条语句、一段程序、一个函数等。在流程图中，模块用一个矩形框表示，如图 5-1 所示。模块的基本特征是其仅有一个入口和一个出口，即要执行该模块的功能，只能从该模块的入口处开始执行（用图 5-1 矩形上面的有向线段表示），执行完该模块的功能后，从模块的出口转而执行其他模块的功能（图 5-1 矩形下面的有向线段），即使模块中包含多个语句，也不能随意从其他语句开始执行，或提前退出模块。

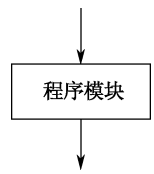


图 5-1 程序模块

5.4.1 程序的三种基本结构

1. 顺序结构

顺序结构是最自然的顺序，由前到后执行，如图 5-2（a）所示。由图 5-2 可以看出，这两个程序模块是顺序执行的，首先执行“程序模块 1”，然后执行“程序模块 2”。

从逻辑上看，顺序结构中的两个程序模块可以合并成一个新的程序模块，如图 5-2（b）所示。通过这种方法，可以将许多顺序执行的语句合并成一个比较大的程序模块。但无论怎样合并，生成的新的程序模块仍然是一个整体，只能从模块的顶部（入口）进入模块开始执行模块中的语句，执行完模块中的所有语句之后再从模块的底部（出口）退出模块。

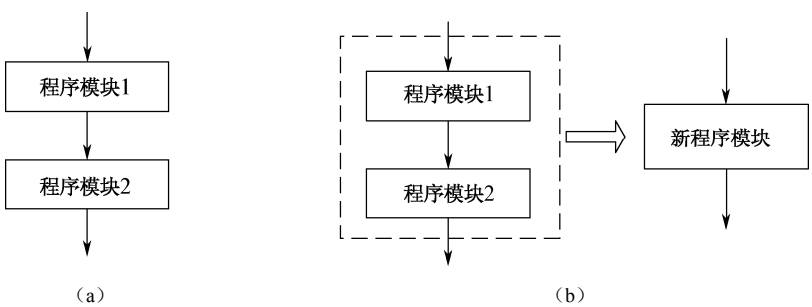


图 5-2 顺序结构

顺序结构是最常见的程序结构形式，在一般程序中大量存在。但是设想一下，是不是所有程序都可以只使用顺序结构编写呢？显然答案是否定的。在求解实际问题时，常常要根据输入数据的实际情况进行逻辑判断，对不同的结果分别进行不同的处理；或者需要反复执行某些程序段落，以避免多次重复编写结构相似的程序段落带来的程序结构上的臃肿。这就需要在程序中引入选择结构和循环结构。一个结构化程序正是由这三种基本程序结构交替综合而构成的。

2．选择结构

选择结构有两种，一种是如图 5-3 所示的结构，这种选择结构的特点是根据逻辑条件成立与否，决定选择执行“模块 1”或者“模块 2”。虽然选择结构比顺序结构稍微复杂了一些，但是仍然可以将其整个作为一个新的程序模块：一个入口（从顶部进入模块开始判断），一个出口（无论执行了“模块 1”还是“模块 2”，都从选择结构框的底部出去）。

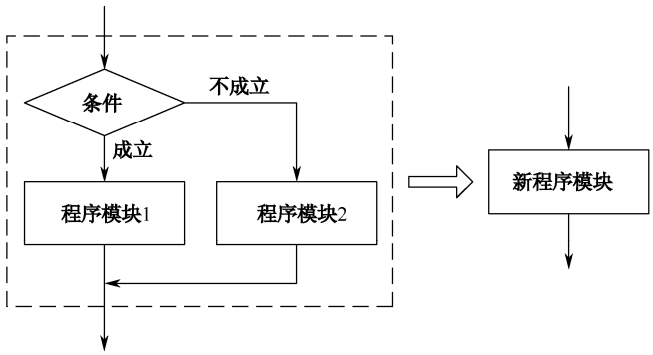


图 5-3　选择结构

在编程实践中，还可能遇到选择结构中的一个分支没有实际操作的情况，如图 5-4 所示。这种形式的选择结构可以看成是图 5-3 所示结构的特例。

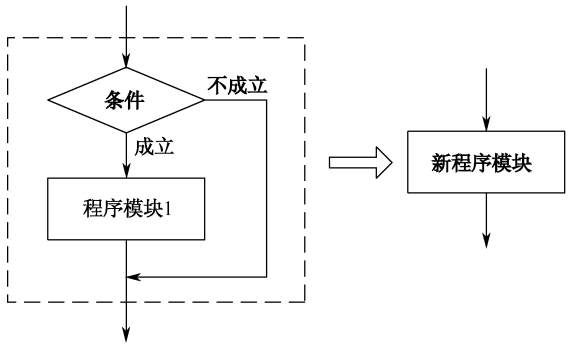


图 5-4　一个分支无实际操作的选择结构

3．循环结构

循环结构也有两种形式，一种是首判断条件是否成立，如果成立则执行“程序模块”，反之则退出循环结构（图 5-5（a））。这种结构的特点是循环体中的“程序模块”可能一次也不会被执行。

另一种循环结构（图 5-5（b）），它是首先执行完“程序模块”，之后再去判断条件是否成立，如果条件仍然成立则再次执行内嵌的“程序模块”，循环往复，直至条件不成立时退出循环结构。这种结构的特点是循环体中的“程序模块”至少会被执行一次。

与顺序和选择结构相同，循环结构也可以抽象为一个新的模块。

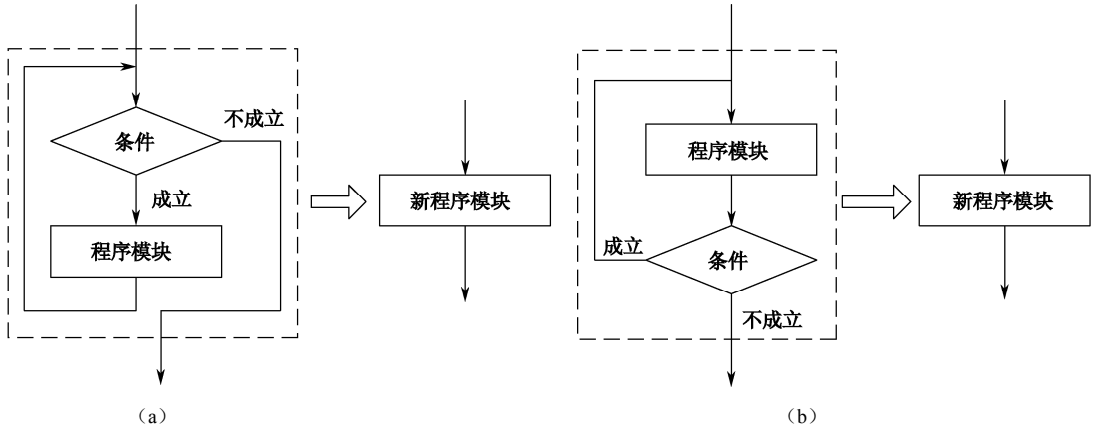


图 5-5 两种循环控制结构

5.4.2 条件分支语句

If-Then 语句用来当条件满足时执行某些语句，反之则不执行。If-Then 语句的格式如下：

```
If <条件表达式> Then
   语句块
End If
```

If-Then 语句又称为单分支结构。其中条件表达式的值为 Boolean 型，用<>将条件表达式括起来表示它在 IF 语句中是必须具有的一项，即 IF 语句中的条件表达式不可缺少。语句块可为一条语句，也可为多条语句。该语句的作用是只有当条件表达式的值为 True 时，程序才执行 Then 后面的语句块。

```
If  <条件表达式> Then
<语句块 1>
Else
<语句块 2>
End If
```

该语句的作用是当条件表达式的值为 True 时，程序执行语句块 1，当条件表达式的值为 False 时，程序将执行语句块 2。

例 5-3 当整数 A＞0 时，将 A 的值存入 B 中。

程序：

```
Dim A As Integer = 45
Dim B As Integer
If A > 0 Then
   B = A
End If
```

例 5-4 找出 A 和 B 中较大的一个数，存入到 C 中。假设 A 和 B 是整数。

程序：

```
Dim A As Integer = 12
Dim B As Integer = 33
Dim C As Integer
If A > B Then
    C = A
Else
    C = B
End If
```

例 5-5　找出 A，B，C 三个数中最大的一个存入到 D 中。

程序：

```
Dim A As Integer = 12
Dim B As Integer = 33
Dim C As Integer = 8
Dim D As Integer
If A > B Then
   If A > C Then
      D = A
   Else
      D = C
   End If
Else
   If B > C Then
      D = B
   Else
      D = C
   End If
End If
```

5.4.3　循环语句

For 循环语句通常用于将一组语句重复执行指定的次数。For 循环的重复次数，可以由设定一个计数变量及其上、下限来决定。语句形式如下（方括号中的内容为可选项）：

```
For 循环变量 = 初值 To 终值 [ Step 步长 ]
   [ 语句块 ]
[ Exit For ]
   [ 语句块 ]
Next [ 循环变量 ]
```

其中，循环变量为必选项。其类型通常是 Integer，但也可以是支持大于号(>)、小于号(<) 和加号 (+) 的任何基本数值类型。每次循环后循环步长的默认值为 1。

- 语句块是放在 For 和变量的增量。一般为正，初值小于终值；若为负，这时初值大于终值；如果省略了该项，则 Next 之间的一条或多条语句，也被称为循环体。它们将被执行指定的次数。
- Exit For，当遇到该语句时，退出循环（无论是否执行完指定次数），执行 Next 语句后面的语句。

例 5-6 计算 1+2+…+100 结果存入变量 sum 中。

分析：使用 For 循环完成。

程序：

```
Dim sum As Integer
For i As Integer = 1 To 100
    sum = sum + i
Next
```

While...End While 循环用于对一条件表达式进行计算，如果值为 True，则执行循环体。每一次循环结束后，重新计算条件表达式。 While...End While 与 For 循环最大的差别在于 For 循环的循环次数是不变的，执行一定次数后结束循环。While 循环的循环次数依赖与条件表达式的值，在不同情况下循环次数不一样。While 型循环结构使用 While 语句实现，语句形式如下：

```
While  <条件表达式>
    [语句块]
End While
```

其中，条件表达式的值必须为 True 或 False。如果表达式的值为 Nothing，则结果将作为 False 处理。

当表达式的值 True 时，则执行 While 后的循环体直到遇到 End While 语句。随后控制返回到 While 语句并再次检查表达式结果。如果表达式仍为 True，则重复上面的过程。如果为 False，则从 End While 语句后面的语句开始执行。

在设计 While 型循环时要注意在其循环体内应该在适当的时候使条件表达式的值为 False，确保在执行了一定次数之后可以退出循环，否则就成了“死循环”，一旦程序进入这里，将永远在循环结构中反复执行而无法结束。

例 5-7 计算 1+1/2+1/3+…+1/n 的值，当 1/n 小于 1e–6 时结束计算。

分析：使用 While 循环完成。

程序：

```
Dim sum As Integer
Dim n As Integer = 1
While 1/n > 0.000001
    sum = sum + 1/n
    n += 1     '注释：等同于 n=n+1
End While
```

例 5-8 体重指数计算。体重指数的计算公式为

BMI=体重（千克）/身高（米）^2

当 BMI＜18.5 时，属于偏轻；在 18.5～24.9 之间时正常；在 25～29.9 之间为偏重；30 以上时肥胖。编写程序，读入输入的体重和身高，计算 BMI 的值并显示相应信息。

程序：

```
'下面的程序使用的控制台输入与输出的代码，见 5.7 节
Dim weight As Single
Console.WriteLine("请输入体重（千克）")
weight = Console.ReadLine()
Dim stature As Single
```

```
Console.WriteLine("请输入身高（米）")
stature = Console.ReadLine()
Dim BMI As Single = weight/(stature ^ 2)
Console.Write("您的体重指数是")
Console.Write(BMI)
Console.Write("体型属于")
If BMI < 18.5 Then
   Console.WriteLine("偏轻")
Else
   If BMI >= 18.5 And BMI <= 24.9 Then
      Console.WriteLine("正常")
   Else
      If BMI > 24.9 And BMI <= 29.9 Then
         Console.WriteLine("偏重")
      Else
         Console.WriteLine("肥胖")
      End If
   End If
End If
```

5.5 数组

在实际应用中，经常需要处理具有相同性质的一批数据。例如，要处理 100 个学生的考试成绩，如果要使用简单变量，将需要声明 100 个变量，极不方便。为此，在 Visual Basic 中，除简单变量外，还引入了数组，即用一个变量表示一组相同性质的数据。每个数据称为数组元素，各元素通过下标来区分。数组必须先声明后使用。在声明数组时，如果数组的大小（包含数据的个数）已经确定，称为静态数组，否则称为动态数组。

如果用一个下标就能确定数组中的不同元素，这种数组称为一维数组，否则称为多维数组。一维数组声明的格式为

```
Dim  数组名(最大下标) As 类型名
```

其中，最小下标为 0。“类型”指定每个元素的数据类型，如 Integer 表明数组中的每个元素都是整型数。Dim 语句声明数组为系统提供了一系列信息，如数组名、数组中各元素的类型、数组的维数和各维的大小等。例如，要表示 10 个学生的成绩，可以声明具有 10 个元素的数组 score，其声明如下：

```
Dim  score(9)  As  Integer
```

该声明表示数组的名字为 score，每个元素为整型数，下标范围是 0～9。各元素通过不同的下标来区分，分别为 score(0)，score(1)，score(2)，…，score(9)。

可以在声明数组时给数组中的元素赋值。例如：

```
Dim s() As Integer = {0,1,2,3,4}
```

此时可以不用指定数组的最大下标，有后面赋值的个数系统自动计算出最大下标。需要注意的是，这样的写法只能在声明数组的同时使用。如果不是这样需要为数组中的元素赋值，则需要为每个数组元素使用赋值语句，通常可以使用一个循环来完成。

例 5-9 假设数组中存有 6 名学生的分数，将每个学生的分数乘以 0.8 后加 20 重新存入到数组中。

分析：开始的成绩在声明数组中给出，随后用一个循环处理每一个元素。

程序：

```
Dim s() As Integer = {98, 67, 78, 99, 87, 82}
For i As Integer = 0 To 5
   s(i) = s(i) * 0.8 + 20
Next
```

也可以使用自定义的数据类型来声明数组，如用 3 个点的坐标来表示平面上的一段折线：

```
Structure Point
       Dim x As Single
       Dim y As Single
End Structure

Dim Line(2) As point
Line(0).x=0
Line(0).y=0
Line(1).x=15
Line(1).y=0
Line(2).x=0
Line(2).y=34.6
```

也可以在申明数组时不指定大小，在使用前用 ReDim 语句重新给定大小。这样的数组称为动态数组。如果要在一个结构的定义中包含数组，则该数组必须是动态的。例如：

```
Dim A() As Integer
ReDim A(100)
```

最后，大多数情形下都要通过一个 For 循环来操作数组中的每一个元素。而 Visual Basic 还提供了一个 For 循环的遍体 For Each…Next 循环来简化这种操作。使用 For Each 循环可以不用监视数组的上限。例如，要取得数组 A 中（假设 A 是一个整形数组）的每一个元素，可以用如下的代码：

```
For Each k As integer in A
    ……
Next
```

该循环依次将数组中的每一个数取到 k 中进行操作（如显示数组中的值）。For Each 循环不但对数组有效，对于大多数存放一批数据的结构，很多也支持 For Each 循环，如第 6 章将要讲到的线性表。

例 5-10 编写一个程序，模拟掷 2 枚骰子的过程。使用 Rnd 随机函数及 Randomize 初始化随机数发生器（请自行查看帮助文件了解如何产生随机数）模拟骰子的情况。每个骰子都有可能产生 1～6 之间的一个整数，因此，2 枚骰子的和在 2～12 之间变化。程序投掷这 2 枚骰子 36 000 次，使用一个一维数组记录每种和出现的次数。

分析：一共有 11 种和的可能，使用数组在第 1 位记录和为 2 的次数，在第 2 位记录和为 3 的次数，以此类推。

程序：

```
Sub Main()
       Dim count(10) As Integer
       Const upperbound As Integer = 6
       Const lowerbound As Integer = 1
       Randomize()
       Dim randomValue1 As Integer
       Dim randomValue2 As Integer
       For i As Integer = 1 To 36000
            '一条语句可以分多行写，在行尾使用下划线表示续行
          randomValue1 = CInt(Math.Floor((upperbound - lowerbound _
                         + 1) * Rnd())) + lowerbound
          randomValue2 = CInt(Math.Floor((upperbound - lowerbound _
                         + 1) * Rnd())) + lowerbound
          count(randomValue1 + randomValue2 - 2) += 1
       Next
       For i As Integer = 0 To 10
          Console.WriteLine("和为{0}的次数为{1}: ", i + 2, count(i))
       Next

   End Sub
```

5.6 子程序过程与函数过程

5.6.1 过程

有些情况下，程序的不同部分可能要执行一段相同的程序代码，那么可以将这一段代码抽出来，建立一个独立的过程，供过程调用。

在 Visual Basic 中，过程分为两类：子程序过程和函数过程，前者又称为 Sub 过程，后者称为 Function 过程。

Sub 过程即子程序，是由 Sub... End Sub 定义的过程。定义过程的格式如下：

```
Sub  过程名（[参数表]）
   语句序列
End  Sub
```

- Sub 过程定义以 Sub 开始，以 End Sub 结束，中间是描述过程功能的语句序列，称为过程体。
- 过程名与变量的命名规则相同，不要与 Visual Basic 中的关键字重名，也不要与同级的变量同名。
- Sub 过程不能嵌套，即不能在 Sub 过程中再定义其他的 Sub 或 Function 过程。

参数表指定在调用该过程时，应该传递的参数的个数和类型。参数表中可以包含多个参数项，相邻的两个参数项之间用逗号隔开。每个参数项的形式如下：

```
ByVal | ByRef  参数名[( ) ]  [ As  类型]
```

这里的参数名是一个合法的变量名，如果参数作为数组来使用，在其名的后面应该加

一对括号。

类型指定参数的数据类型，可以是 Integer，Long，Single，Double，String，Variant 或用户定义的其他类型。如果省略“As 类型名”则默认为 Object。变量名前的 ByVal 表明参数是按照值来传递，ByRef 按照地址来传递。有关细节，本章后面将做详细介绍。这里的参数在定义时并没有分配存储单元，只有在运行该过程时才分配，所以又称为形式参数。

End Sub 表明过程的结束。每个 Sub 过程应该有一个 End Sub 语句，当程序执行到 End Sub 时，将结束该过程的运行。在过程体中，还可以包含一个或多个 Exit Sub 语句，当程序运行到 Exit Sub 语句时，也结束过程的运行。

5.6.2 调用 Sub 过程

调用 Sub 过程，即执行该过程中的代码。调用 Sub 过程的形式如下：

```
过程名([实际参数表])
```

其功能是运行该过程名对应的过程。实际参数表是传递给该过程的诸参数，实际参数简称实参。实参可以是常量、变量或表达式。相邻的两个实参之间用逗号隔开。实参的个数、顺序、类型和形参要一一对应。

调用的执行过程是，首先将实参传递给形参，然后执行过程体。当过程运行结束后，从调用该过程的语句的下一句处继续执行。例如，要调用过程 area，使用如下的程序段：

```
Dim x As Integer
Dim y As Integer
x = 5
y = 7
area(x, y)
```

当执行该程序段时，首先给变量 x 和 y 赋值，接着以 x 和 y 为实参，调用过程 area。

调用过程是首先将实参 x 传递给形参 a，将实参 y 传递给形参 b，然后执行过程 area 的过程体，即该过程中的诸语句。当执行到 End Sub 时，过程的运行结束，返回到调用该过程的语句的下一句，从该处继续程序的运行。

5.6.3 Function 过程

Sub 过程不返回值，且以语句的形式调用。Function 过程要返回一个值，调用方式是以表达式的形式出现。Function 过程的定义格式如下：

```
Function 过程名 ([参数表]) [As 类型名 ]
语句序列
End Function
```

Function 过程以 Function 开始，以 End Function 结束，中间是描述过程功能的语句序列，称为过程体或函数体。

过程体中至少有一条 Return 语句，形式如下：

```
Return 表达式
```

当调用该过程时，过程的返回值即为表达式的值。“As 类型名”指定 Function 过程返回值的数据类型。其他部分同 Sub 过程的定义。例如，定义计算阶乘的 Function 过程如下：

```
Function facts( n As Integer) As Long
    Dim i As Integer
```

```
    Dim result As Long
    result = 1
    For i = 1 To n
        result = result * i
    Next
    Return result
End  Function
```

该函数过程包含一个 Integer 型的形参，其返回值为 Long 型。可以看到，过程体中包含 Return 语言，返回值等于参数 n 的阶乘。

5.6.4 Function 过程的调用

由于 Function 过程返回一个值，可以像其他函数一样来调用。将它作为单独的语句来调用，就无法得到函数的返回值。因此，一般它作为表达式或表达式的一部分出现。其在表达式中出现的形式如下：

```
过程名（[参数表]）
```

参数表的使用形式同调用 Sub 过程。例如，要调用前面定义的计算阶乘的 Function 过程 facts，可以采用下面的程序段：

```
Dim  m  As  Integer = 12
Dim y As Long
y = facts(m)
```

当执行到 y = facts(n)时，将调用函数过程 facts。首先将实参 m 传递给形参 n，然后执行该函数过程的过程体，当执行到 End Function 时，完成过程的运行，将 Return 后面表达式的值带回调用处，即赋给变量 y，然后从调用处继续执行。

5.6.5 参数传递

在调用一个过程时，必须为过程提供实际参数，完成实际参数与形式参数的结合，称为参数传递，然后用实际参数执行所调用的过程体。

在 Visual Basic 中，参数传递的方式有两种：传值和传址。传值是将实际参数的值传递给形式参数，而传址是将实际参数的引用（地址）传递给形式参数。

如果要按照传址方式进行参数传递，在定义过程时，在形式参数的前面加 ByVal；如果按照传址方式进行参数传递，在形式参数的前面加 ByRef。

参数传递的传值方式是将实际参数的值传递给形式参数，而不是将实际参数的地址传递给形式参数。在这种方式下，系统将要传送的变量复制到一个临时单元中，然后把临时单元的地址传送给被调用的过程，即系统为形式参数重新分配存储单元。由于被调用过程没有访问实际参数，因而在改变形式参数值时，并没有改变实际参数的值。

要使得参数的传递按照传值方式进行，需在定义过程时，在形式参数的前面加 ByVal 关键字。

参数传递的另一种方式是引用，又称为传地址。定义过程时，如果在形式参数的前面加 ByRef，参数的传递方式是引用。

在引用方式下，参数的传递是将实际参数的地址传递给形式参数，所以，形式参数和实际参数共享相同的存储单元。当在过程中对形式参数的值更改时，实际参数的值也进行

相应的更改。所以，可以通过引用的方式，将被调用过程的处理结果带回调用过程。

定义过程时，是选择传值参数还是选择引用方式，没有统一的标准，下面几条原则可供参考：

- 如果要将被调过程中的结果带回，采用引用参数，否则采用传值参数。
- 对于整型、长整型或单精度型参数，如果不希望过程修改实参的值，采用传值参数；反之可使用引用参数。

前面介绍过，Function 过程能通过返回值直接带回过程的结果，而 Sub 过程由于没有返回值，不能直接带回过程的结果。但是通过传址方式的参数，Sub 过程也能间接带回过程的结果。所以在编写程序时，通过 Function 过程能解决的问题，同样也能通过 Sub 过程完成，反之亦然。不过，如果过程的结果要以表达式的形式使用，采用 Function 过程为佳。

例如，要计算 n!，既可以使用 Function 过程，也可以使用 Sub 过程。使用如下 Function 过程：

```
Function  Facts(ByVal n As Integer) As Long
    Dim i As Integer
    Dim s As Long = 1
    For i = 1 To n
        s = s * i
    Next
    Return s
End  Function
```

使用如下 Sub 过程：

```
Sub  Facts(ByVal n As Integer, ByRef y As Long)
    Dim i As Integer
    y = 1
    For  i = 1 To n
          y = y * i
          Next
End  Sub
```

5.6.6 值变量和引用变量与参数传递

Visual Basic 中的变量可以分为两类：值类型和引用类型。如果某个数据类型在自己的内存分配中包含数据，则该数据类型是“值类型”。“引用类型”含有指向包含数据的其他内存位置的地址（指针）。以下变量是值类型：

- 所有 numeric 数据类型。
- Boolean、Char 和 Date。
- 所有结构（自定义数据类型），即使其成员是引用类型。

以下变量是引用类型：

- String。
- 所有数组，即使其元素是值类型。
- 类类型。

对于值类型，前面已经看到，在数或过程中如果通过 ByVal 方式传递参数，则在函数和过程中对形参做的任何改变不会影响实参的值。如果通过 ByRef 传递参数，则对形参的

改变将导致实参同时改变。

对于引用类型，则有所不同。在引用类型的变量中，本身就是存放的变量的地址。因此，无论是通过 ByVal 和 ByRef 方式传递参数，在函数或过程中，对变量中内容的改变都将导致调用代码中实际参数的改变。区别在于，以 ByVal 方式传递参数时，只能改变变量的内容，变量本身的改变不会影响实际参数。以 ByRef 方式传递参数时，既可以改变变量的内容，也可以改变变量本身。例如，以 ByVal 方式传递数组参数时，改变数组中的内容时，调用代码中的实际数组内容也会改变。但是如果在函数或过程中重定义数组的大小，则调用代码中的数组不会改变。如果以 ByRef 方式传递的数组，则这两者都会跟着改变。

5.6.7 Sub Main

Visual Basic 代码一般都应该在一个过程中。因此，Visual Basic 至少要有一个过程，该过程的名字是指定的，即 Sub Main() 过程。程序开始运行时将从 Main 过程的第一条语句开始执行，执行完 Main 过程的最后一条语句，整个程序运行结束。使用 Visual Studio 2008 编写 Visual Basic 程序时，Visual Studio 2008 将自动写好 Sub Main 和 End Sub 语句（见 5.8.1 节图 5-7）。如果程序不需要其他的过程，则只需要将代码填入 Sub Main 过程即可。也就是说，在前面的例 5-1～例 5-10 中，所有的代码应该在 Sub Main 过程中。

5.6.8 变量的作用范围

变量是有作用范围的，在一个过程中声明的变量在其他过程中则是不可访问的。或者，换一个角度说，在两个不同的过程中声明相同名字的变量，这两个变量是相互独立，没有什么关联的。更进一步，如在一个循环体中声名的变量，则只在这个循环体中是有效的，出了循环体，该变量就不可访问了，或者说被销毁了。

例如，有以下的程序片段：

```
For i As Integer=0 To 9
    Dim B As Integer
    B=i
Next
    Dim A As Integer
    A=B  '错误的语句
```

在以上程序段中，最后一句是错误的，原因是 B 是在 For 循环中声明的，当退出 For 循环时，变量 B 同时也会被销毁，因此，最后一句 A=B 时，B 已经不存在了，所以是错误的。如果确实要使用 B，可以将 B 的声明放到 For 循环的外面。同样的道理，也适用于过程。过程中的参数，被看作是在过程内声明的变量。

例 5-11 计算两个矩形面积之和。矩形的面积使用一个 Function 过程来完成。

分析：假定两个矩形的边长分别为 3.4，6.7 和 4.5，8.9。

程序：

```
Sub Main()
   Dim x1 As Decimal = 3.4
   Dim y1 As Decimal = 6.7
   Dim x2 As Decimal = 4.5
   Dim y2 As Decimal = 8.9
```

```
    Dim Sum As Decimal
    Sum = Area(x1, y1) + Area(x2, y2)
End Sub

Function Area(ByVal x As Decimal, ByVal y As Decimal)
    Return x * y
End Function
```

例 5-12 使用 Sub 过程，完成例 5-9 同样的任务。

分析：Sub 函数没有返回值，因此，需要有额外的参数通过 BayRef 的方式将计算结果带回到调用函数。

程序：

```
Sub Main()
    Dim x1 As Decimal = 3.4
    Dim y1 As Decimal = 6.7
    Dim x2 As Decimal = 4.5
    Dim y2 As Decimal = 8.9
    Dim Sum As Decimal
    Dim A1 As Decimal
    Dim A2 As Decimal
    Area(x1, y1, A1)
    Area(x2, y2, A2)
    Sum = A1 + A2
End Sub

Sub Area(ByVal x As Decimal, ByVal y As Decimal, ByRef A As Decimal)
    A = x * y
End Sub
```

*5.6.9 递归调用

在数学上，关于递归函数的定义是指对于某一函数 f(x)，其定义域是集合 A，那么若对于 A 集合中的某一个值 x0，其函数值 f(x0)由 f(f(x0))决定，那么就称 f(x)为递归函数。在编程语言中，把直接或间接地调用自身的函数称为递归函数。这样的递归函数通常必须满足以下两个条件：

- 在每一次调用自己时，必须是（在某种意义上）更接近于解。
- 必须有一个终止处理或计算的准则。

例 5-13 使用递归函数计算 n 的阶乘。

分析：阶乘的递归可以定义为（1）n!=n*(n-1)!。（2）当 n=0 时，0！=1。（1）指出了递归的定义，而（2）则给出了结束的条件，二者缺一不可。

程序：

```
Module Module1

    Sub Main()
        Dim a As Integer = Fac(6)
        Console.WriteLine(a)
```

```
    End Sub

    '阶乘的递归函数
    Function Fac(ByVal n As Integer) As Integer
        If n = 0 Then
            Return 1
        Else
            Return n * Fac(n - 1)    '递归调用
        End If
    End Function

End Module
```

5.7 对象和类

我们可能已经听说过面向对象编程（Objecte-Oriented Programming，OOP）这个术语。OOP 最关键的一点就是使用可重用对象来构建程序。面向对象程序设计方法认为，客观世界是由各种各样的实体组成的，这些实体就是面向对象方法中的对象。一般的认为，对象是包含现实世界物体特征的抽象实体，反映了系统为之保存信息和与之交互的能力。

5.7.1 对象

每个对象有各自属性和方法（也就是过程），整个程序是由一系列相互作用的对象构成的，对象之间的交互通过发送消息（调用对象的方法）来实现。对象可视为一个单元的代码和数据的组合。对象可以是一段应用程序，整个应用程序也可以是一个对象。

属性是对象的特性，它们定义对象的特征之一（如大小、颜色或屏幕位置），或者定义对象行为的某一方面（如是否启用或可见）。若要更改对象的特征，可更改其相应属性的值。而方法则是对象的一组过程，主要用来对象的数据进行操作。在程序中，通过对象名加上一个句号后跟属性或方法的形式来访问对象的属性或方法。

事实上，在 Visual Basic 中，前面介绍的数组就是一个对象。例如，下面的程序段：

```
Sub Main()
       Dim A() As Integer = {23, 45, 67, 12, 49}
       Dim L As Integer
       L = A.GetLength(0)
    End Sub
```

数组 A 就是一个对象，A 调用其方法（也就是 GetLength Function 过程得到数组的长度，参数 0 表示是第一维的长度）程序运行后 L 的值为 5。

5.7.2 类

简单的说，类是对象的模板。或者说类与对象的关系是图纸和实体之间的关系。上面的例子中，数组 A 根据一个预先定义好的类（Array 类）而创建。同样可以创建数组 B 和 C 等。A，B 和 C 都是对象，他们都根据同一个类 Array 而创建。也可以自己先定义一个类，然后再根据这个类创建出 1 个或者多个对象。类使用来定义。

```
Class MyClassName
…
End Class
```

一般而言，创建一个类的对象需要使用 New 关键字，格式如下：

```
Dim classA As New MyClassName()
```

MyClassName 后面的括号，类似于函数参数的传递，一般用于创建对象时给对象中一些变量初值。如果没有，括号内为空。此时括号可以省略，但作者觉得有括号的编程风格更好。

需要注意的是，如果一个类是被申明为静态的，则不需要创建对象，使用类名后跟句号加方法名的形式来调用。例如，Visual Basic 2008（严格的说应该是.NET）预先定义了 Math 类就是这样的类。Math 类中包含了许多与数学运算相关的方法，他们都是静态的。因此，不需要根据 Math 类来创建一个对象，而是直接使用类名。如果要计算 x 的正弦函数的值，写法如下：

```
Math.Sin(x)
```

5.8 控制台的输出与输入

以上简单介绍了 Visual Basic 2008 的本书要用到的一些基本的语法。但是一个实际的程序，总是离不开输入和输出的。严格地讲，输入和输出并不是 Visual Basic 语言的一部分。和前面提到的 Math 类相同，输入和输出依靠预先提供的方法来完成的。这些方法都包含在 Console 类当中。和 Math 类同样，Console 类中的方法都是静态的，因此，不需要创建一个 Console 类的对象（或者称为实例）。

输入和输出对于本书的例子而言，输入是指在程序运行时，从键盘得到要处理的信息。输出是指将结果显示在屏幕上。在此，也只讨论字符界面（控制台的输入和输出）。

控制台是一个操作系统窗口，用户可在其中通过计算机键盘输入文本，并从计算机终端读取文本输出，从而与操作系统或基于文本的控制台应用程序进行交互。例如，在 Windows 中控制台称为命令提示窗口，可以接收 MS-DOS 命令。Console 类对从控制台读取字符并向控制台写入字符的应用程序提供基本支持。Console 类提供用于从控制台读取单个字符或整行的方法；该类还提供若干写入方法，可将值类型的实例、字符数组，以及对象集自动转换为格式化或未格式化的字符串，然后将该字符串写入控制台。

需要注意的是，实际上输入和输出都是针对字符和字符串（或文本）的。因此，输出到控制台时，所有数据都应该转换为字符或字符串。然而，很多时候，这种转换是自动的。我们并不需要提供额外的代码来转换。而输入则没有这种自动的转换，从键盘的输入将作为一个字符或字符串被读取，可能需要额外的代码将其转换为整数、小数或其真正需要的数据类型。

5.8.1 控制台的输出

控制台的输出主要通过 Console.Write 及带有换行的 Console.WriteLine 来完成。下面的程序段向控制台（屏幕）输出整数、字符串等：

```
Dim A As Integer=56
Console.Write(A)
Console.Write("Hello World")
```

Console.Write 和 Console.WriteLine 功能基本是一样的，主要的区别在于 WriteLine 输出完之后会跟一个换行符。下一个输出会从一个新行开始。另外，需要注意的是，Console 类中并非只有一个 Write 方法，而是有多个。只不过这些方法是同名的而已。上面的程序段中，输出整数的和输出字符串的实际是两个不同的 Write 方法。这种现象称为重载。

Write 和 WriteLine 方法还有一个重要的用法是使用占位符的方式。例如：

```
Console.Write("X+Y={0}+{1}={2}",X,Y,X+Y)
```

在上面的语句中，最终输出是 Write 参数的第一个字符串 X+Y={0}+{1}={2}，但是{0}等是占位符，实际输出时，{0}，{1}和{2}的值将被 X，Y 和 X+Y 的值所取代。因此，也要求占位符的个数应该和后面替换的参数个数一致且一一对应。

例 5-14 Console.Write 的输出示例。

程序：

```
Sub Main()
    Dim A As Integer = 23
    Dim B As Integer = 34
    Console.Write("Welcome VB 2008")
    Console.Write("  ")    '输出 2 个空格
    Console.Write(A)
    Console.Write(vbCrLf)    'vbCrLf 是预先定义好的换行符
    Console.Write("A={0},B={1},A+B={2}", A, B, A + B)
End Sub
```

程序的运行结果如图 5-6 所示。

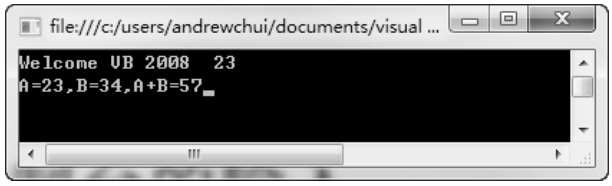


图 5-6 例 5-14 程序的输出，注意换行个占位符的使用

例 5-15 Console.WriteLine 的输出示例。

程序：

```
Sub Main()
    Dim A As Integer = 23
    Dim B As Integer = 34
    Console.WriteLine("Welcome VB 2008")
    Console.WriteLine(A)
    Console.WriteLine("A={0},B={1},A+B={2}", A, B, A + B)
End Sub
```

程序运行的输出如图 5-7 所示。

除了使用{0}作为占位符之外，还可以加入其他的参数，进一步控制输出的格式。例如：

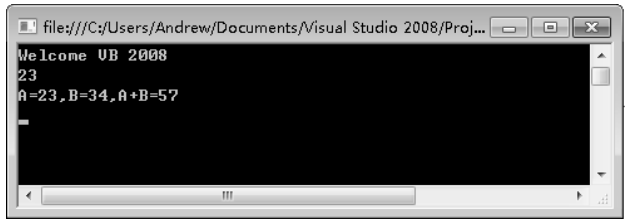


图 5-7 例 5-15 的输出，WriteLine 每次输出一行

```
Sub Main()
    Dim i As Decimal = 123.45678
    Console.WriteLine("{0:C}", i)
    Console.WriteLine("{0:F}", i)
    Console.WriteLine("{0:E}", i)
End Sub
```

第一行将按货币格式输出，第二行按定点格式（小数点后 2 位），第三行按科学计数法输出。本书不在此罗列所有的输出格式的控制了，在以后用到的地方再解释。

5.8.2 控制台的输入

控制台的输入，大多数时候是指从键盘输入。也就是程序在运行时从键盘得到要处理的数据。这主要依靠两个过程 Console.Read 和 Console.ReadLine 来完成。Console.Read 方法读取一个字符，而 Consoel.ReadLine 方法读取一个字符串。

需要注意的是，无论程序最终需要什么类型的数据，在输入时，我们只能将输入作为字符或者字符串读入。读入数据后，程序需要自己将读入的数据转换为需要的数据类型（如果有必要的话）。由于 Read 方法每次只能读入一个字符，因此，在实际使用时 ReadLine 使用的更多一些。

例 5-16 从键盘输入一个字符串到程序中。

程序：

```
Sub Main()
    Dim s As String
    s = Console.ReadLine()
    Console.WriteLine(s)
End Sub
```

上述程序运行后，程序会暂停等待用户从键盘输入。用户输入内容并按下回车键后，输入的内容将被存储到字符串 s 中。随后将 s 的内容显示到屏幕上。假设输入的内容为 Hello VB2008。程序运行的结果如图 5-8 所示。

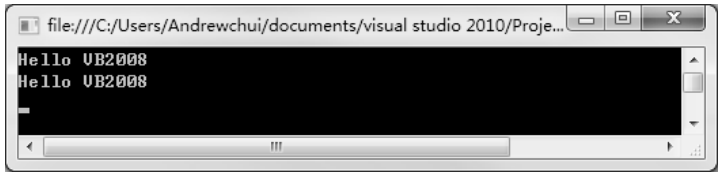


图 5-8 程序运行后等待用户的输入，并将输入的内容读入程序

如果实际是需要输入一个数字呢？例如，输入整数或者是小数，此时该如何？这时，

仍然是将输入的内容作为一个字符串读入，然后有程序将字符串转换为需要的类型。有多种转换的方式，在这里，我们使用类 Convert 提供的方法来转换。和 Console 类相似，Convert 也提供了一组静态的方法来完成转换，因此，使用 Convert 类的方法时也不需要创建对象。常用的转换方式如表 5-6 所示。

表 5-6　常用的 Convert 方法

方法名称	转换后的数据类型
Conver.ToBoolean	Boolean
Convert.ToInt32	Integer
Convert.ToInt64	Long
Conver.ToDecimal	Decimal
Convert.ToSingle	Single
Convert.ToDouble	Double
Convert.ToChar	Char
Convert.ToDateTime	DateTime

例 5-17　从键盘输入两个小数，相加后将显示结果。

分析：读入数据，并将数据类型转换后相加并显示结果。

程序：

```
Sub Main()
    Dim s As String
    Dim A As Decimal
    Dim B As Decimal
    s = Console.ReadLine()
    A = Convert.ToDecimal(s)
    s = Console.ReadLine()
    B = Convert.ToDecimal(s)
    Dim C As Decimal = A + B
    Console.WriteLine("{0}+{1}={2}", A, B, C)
End Sub
```

程序运行时，先将数据读入到 s 中，再把字符串转换为小数。每次输入一个数时，需要按下回车键，即一行输入一个数据。程序运行的结果如图 5-9 所示。

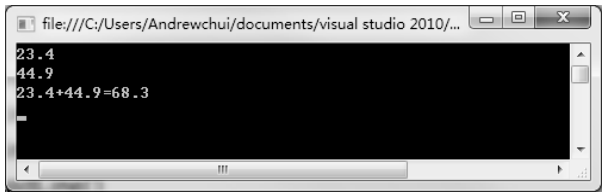


图 5-9　将输入的数据转换后相加，注意一行输入一个数据

上面的例子中先将 23.4 作为字符串"23.4"存储到了变量 s 中，然后再转换为小数存储到 A 中。要注意需要确保这样的转换是可行的，如果试图将字符串"ABC"转换为小数程序会立即中止运行并报错。

如果想在一行内输入多个数据，则需要做更多的工作。首先，这些数据会作为 1 个字符串读入，当采用空格来分割时，形如"12 45 23 56 88"。此时在一行内输入了 5 个整数，并以空格分隔的。接下来需要将这个字符串分解为 5 个字符串，存入到一个字符串数组中。

分解的依据是空格作为分隔符。可以使用字符串对象自己的一个方法 Split 来完成此事。然后使用循环依次转换。

例 5-18 在一行内输入多个数据。在一行内输入 5 个整数，分别求其除 3 的余数。

分析：将输入的一行字符串分割为多个字符串，分别转换。

程序：

```
Sub Main()
    Console.WriteLine("请输入 5 个整数,以空格分隔并以回车结束")
    Dim s As String = Console.ReadLine()
    Dim sSplit() As String = s.Split(" ")
    Dim A As Integer
    For i As Integer = 0 To 4
       A = Convert.ToInt32(sSplit(i))
       Console.Write((A Mod 3).ToString() + " ")
    Next
    Console.WriteLine()
End Sub
```

上面的程序中，ReadLine 将空格分隔的 5 个整数作为一个字符串读入到变量 s 中。随后调用变量 s（也是一个数组对象）的 Split 方法将 s 按空格分割成 5 个字符串结果存入字符串数组 sSplit 中。在输出的 Write 方法中，A Mod 3 的结果是一个整数，整数也是一个对象，用 ToString 方法把自己转换为一个字符串，在和含有一个空格的字符串连接后输出。假设输入的 5 个整数是 5、6、7、8 和 9。程序的运行结果如图 5-10 所示。

图 5-10 在一行内输入多个数据的示例

5.9 使用 Visual Studio 2008

Visual Studio 是微软公司推出的开发环境，是目前最流行的 Windows 平台应用程序开发环境。我们只会用到其中的一小部分功能。下面通过一个例子来说明。

5.9.1 控制台应用程序的创建与运行

例 5-19 跳水比赛采用 7 名裁判评分，分数在 0～10 之间。评分是，去掉一个最高分和一个最低分，剩下的分数相加后乘以 3 再除 5，然后乘上难度系数为该动作的最后得分。编写程序，输入以上数据，计算运动员的分数。

步骤：

（1）运行 Visual Studio 2008，从菜单选择新建→项目。在弹出的对话框选择项目类型为 Visual Basic→Windows；模板选择控制台应用程序；名称一栏输入 Diving，如图 5-11 所示。

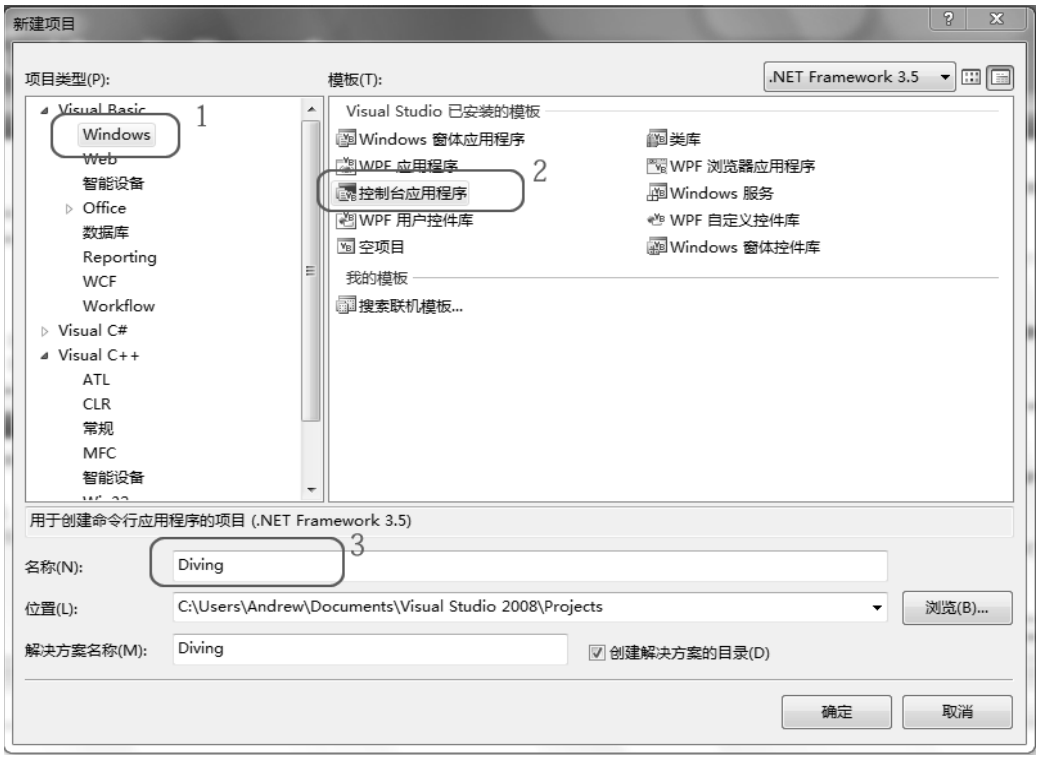


图 5-11　新建 Visual Basic 控制台应用程序

（2）在新建的项目中输入以下代码（灰底部分是实际需要输入的，图 5-12）：

```
Module Module1

    Sub Main()
        Dim difficult As Decimal = 3.1   '定义一个小数,跳水的难度系数
        Dim Score() As Decimal = {8.0, 8.5, 8.0, 8.5, 9.0, 9.5, 8.0}
                                         '定义一个数组,打分
        Dim max As Decimal = Score(0)
        Dim min = Score(0)           'min 的类型 AS Decimal 可以省略，由系统决定
        Dim Sum = Score(0)
        For i As Integer = 1 To 6
            If Score(i) > max Then
                max = Score(i)         '找出得分的最大值
            End If
            If Score(i) < min Then
                min = Score(i)         '最小值
            End If
            Sum = Sum + Score(i)
        Next
        Sum = (Sum - max - min) * 3/5 * difficult
        Console.WriteLine("运动员的得分为{0,1:F}", Sum)    '输出
    End Sub

End Module
```

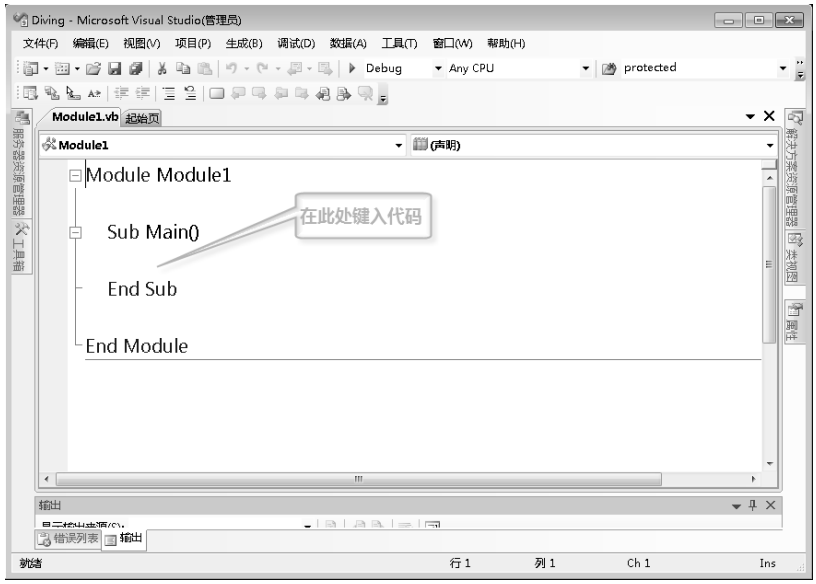


图 5-12 键入代码的位置

（3）运行程序。从调试菜单→开始执行（不调试）得到运行结果如图 5-13 所示。

图 5-13 程序的运行结果

（4）如果想从键盘输入数据来运行程序的话，代码如下：

```
Module Module1
Sub Main()
    Dim difficult As Decimal
    Console.WriteLine("请输入难度系数,以回车结束")
    difficult = Convert.ToDecimal(Console.ReadLine())
    Console.WriteLine("请输入 7 名裁判的评分,以空格分隔并以回车结束")
    Dim strScore As String = Console.ReadLine()
    Dim strSco() As String = strScore.Split(" ")
    Dim score(6) As Decimal
    score(0) = Convert.ToDecimal(strSco(0))
    Dim max As Decimal = score(0)
    Dim min As Decimal = score(0)
    Dim sum As Decimal = score(0)
    For i As Integer = 1 To 6
        score(i) = Convert.ToDecimal(strSco(i))
        If score(i) > max Then
            max = score(i)
        End If
        If score(i) < min Then
```

```
                min = score(i)
            End If
            sum = sum + score(i)
        Next
        sum = (sum - max - min) * 5/3 * difficult
        Console.WriteLine("运动员的得分为{0,1:F}", sum)
    End Sub
End Module
```

程序的运行结果如图 5-14 所示。

图 5-14 从键盘输入数据时，程序的运行结果

5.9.2 Visual studio 2008 集成环境

Visual Studio 2008 启动后，会看到如图 5-15 的程序界面。要开始编写一个新的程序，首先需要建立一个解决方案，该解决方案里包含一个项目。简单起见，可以直接创建项目，Visual Studio 将自动为该项目创建一个解决方案。注意：图 5-15 的启动界面是在第一次启动 Visual Studio 后选择了常规开发设置后的界面，如果选择了其他设置，界面也许会有所区别。

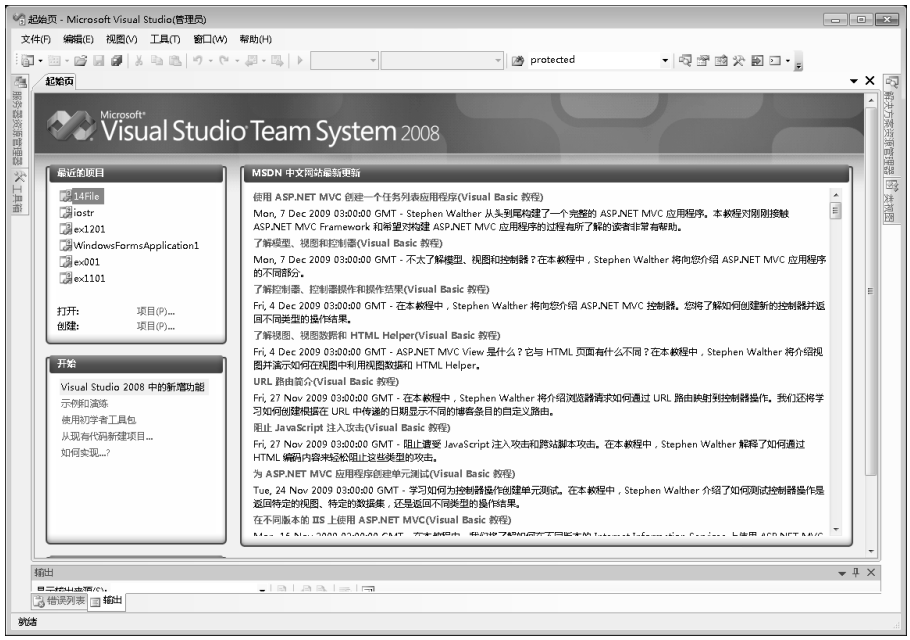


图 5-15 Visual Studio 2008 的启动界面

编写程序的第一步应该是在文件菜单中选择新建→项目。随后会看到图 5-11 的界面。

输入完代码后，如果有语法错误，会在代码窗口中用波浪线指出（图 5-16）。在生成菜单选择生成解决方案（意思是将源代码编译成机器码），如果有错误则会在输出窗口显示错误，同时错误也显示在错误列表窗口。

图 5-16 输入代码并改正语法错误

修改完语法错误的程序便可以进入调试阶段，看看程序在运行时是否正确。可以设置断点使程序在运行中暂时停止，可以查看程序中变量的值，也可以单步执行程序，如图 5-17 所示。

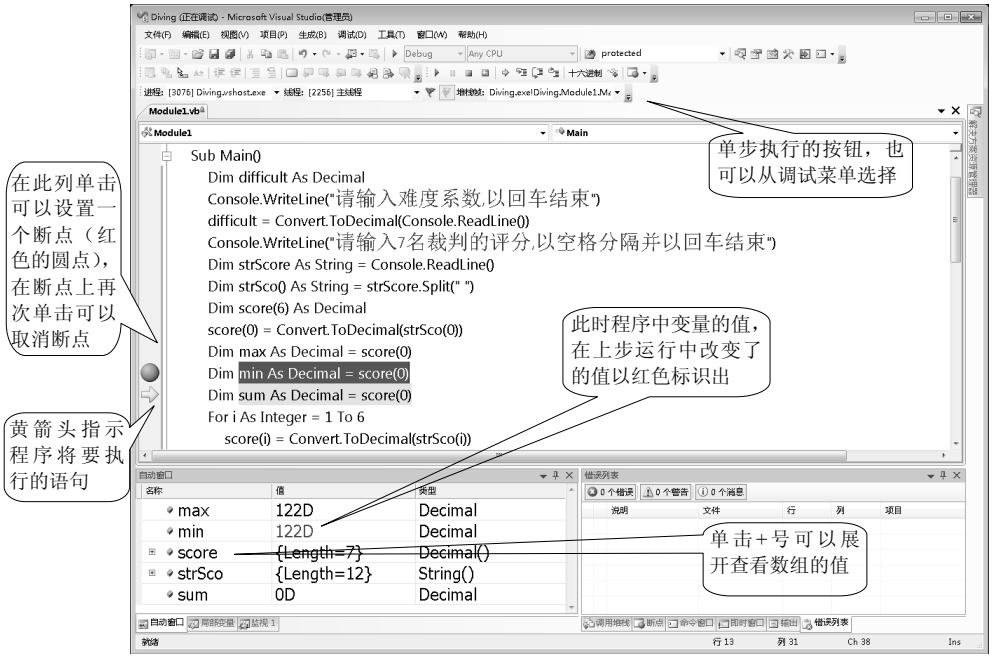


图 5-17 程序的运行与调试

5.10 范例程序阅读

例 5-20 平面上已知坐标的任意三个点 A(x1,y1)，B(x2,y2)，C(x3,y3)（3 个点的坐标在程序中直接给定），检验它们能否构成三角形。若不能，则输出相应的信息；若能，则输出三角形的面积和周长。

分析：已知两个点的坐标，可以计算出两个点之间的距离，计算公式为

$$\text{Distance} = \sqrt{(x1-x2)^2+(y1-y2)^2}$$

3 个点总共可以组成 3 条边，每条边的长度可以由上式计算得到。组成三角形的条件是两边之和大于第三边，只要满足该条件则是三角形，否则就不是三角形。已知三角形的三条边的长度，根据海伦公式就可以计算出该三角形的面积，海伦公式为

$$\text{area} = \sqrt{s(s-a)(s-b)(s-c)}$$

其中，$s = \dfrac{a+b+c}{2}$，area 为面积。

在求解过程中需要使用开平方数学函数，使用 Math.sqrt(x)来完成。

程序：

```
Module Module1

    Sub Main()
        '定义 6 个实数来存储坐标点
        Dim x1 As Double = 2.1
        Dim x2 As Double = 0
        Dim x3 As Double = 4.3
        Dim y1 As Double = 0.7
        Dim y2 As Double = 0
        Dim y3 As Double = 9.2
        '计算 3 边长
        Dim d1 As Double = Math.Sqrt((x1 - x2) ^ 2 + (y1 - y2) ^ 2)
        Dim d2 As Double = Math.Sqrt((x1 - x3) ^ 2 + (y1 - y3) ^ 2)
        Dim d3 As Double = Math.Sqrt((x2 - x3) ^ 2 + (y2 - y3) ^ 2)
        '判断是否可以组成三角形
        If d1 + d2 > d3 And d2 + d3 > d1 And d1 + d3 > d2 Then
            '如果可以组成三角形，则计算面积和周长并输出
            Dim s As Double = (d1 + d2 + d3) / 2
            Dim area As Double = Math.Sqrt(s * (s - d1) * (s - d2) * (s - d3))
            Console.WriteLine("可以组成三角形")
            Console.WriteLine("三角形的面积是{0}", area)
            Console.WriteLine("三角形的周长是{0}", s * 2)
        Else
            Console.WriteLine("无法组成三角形")
        End If
    End Sub

End Module
```

例 5-21 一对兔子，从出生后第 3 个月起每个月都生一对兔子。小兔子长到第 3 个月后每个月又生一对兔子。假如兔子都不死，请问第 1 个月出生的一对兔子，至少需要繁衍到第几个月时兔子总数才可以达到 R 对？（R 为给定的正整数，从键盘输入）。

分析：设某一个月的当月出生的兔子有 R1 对，两个月大的兔子有 R2 对，3 个月及以上大小的兔子有 R3 对，则总数是 R1+R2+R3 对。那么再过 1 个月后，3 个月及以上的兔子数量是 R3+R2，2 个月的是 R1 对，刚出生的是 R3 对。第 1 个月的初始值：R1=1、R2=0、R3=0。把每个月的各月龄兔子数量序列都打印出来，当超过 R 时得到月份数。正确的序列是 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, 144, 233, 377, 610, 987, 1597……

程序：

```
Module Module1

    Sub Main()
        Dim month As Integer = 1
        Console.WriteLine("输入兔子总的数量")
        Dim R As Integer
        R = Convert.ToInt32(Console.ReadLine())
        Dim R1 As Integer = 1
        Dim R2 As Integer = 0
        Dim R3 As Integer = 0
        While R1 + R2 + R3 < R
            month += 1        'month=month+1
            R3 += R2          '二月龄兔子变成成年兔子
            R2 = R1           '一月龄兔子变成二月龄兔子
            R1 = R3           '成年兔子又生出一月龄兔子
        End While

        Console.WriteLine("至少第{0}个月才有{1}对兔子", month, R1 + R2 + R3)
    End Sub

End Module
```

例 5-22 输入两个整数 A 和 B，输出从 A 到 B 的所有整数及这些数的和。如输入 A=–3，B=8 则总合为（–3）+（–2）+…+8=30。

程序：

```
Module Module1

    Sub Main()
        Dim A As Integer
        Dim B As Integer
        Console.WriteLine("请输入 A 的值")
        A = Convert.ToInt32(Console.ReadLine())
        Console.WriteLine("请输入 B 的值")
        B = Convert.ToInt32(Console.ReadLine())
        Dim sum As Integer
        For i As Integer = A To B
            sum = sum + i
```

```
        Next
        Console.WriteLine("从A到B的值是{0}", sum)
    End Sub

End Module
```

例 5-23 编写一个函数 IsPrime 来判断一个数是否为素数，若是则函数值为真，否则为假。输出 2～*n* 之内的所有素数。

分析：对于一个指定的数 n，从 2 开始，直到 Sqrt(n)，如果都无法整除 n 则可以判定 n 是一个素数。

程序：

```
Module Module1

    Sub Main()
        Dim n As Integer
        Console.WriteLine("请输入n的值")
        n = Convert.ToInt32(Console.ReadLine())
        If n < 2 Then
            Console.WriteLine("输入错误")
            End    'End 语句立即结束程序的运行
        End If
        For i As Integer = 2 To n
            If IsPrime(i) Then
                Console.Write(i)
                Console.Write(" ")
            End If
        Next
        Console.WriteLine()
    End Sub

    Function IsPrime(ByVal n As Integer) As Boolean
        If n = 2 Then
            Return True
        Else
            Dim i As Integer = 2
            While i <= n/i
                If n Mod i = 0 Then
                    Return False       '整除不是素数
                End If
                i += 1
            End While
            Return True
        End If
    End Function

End Module
```

*5.11 关于 Visual Basic 2008 其他应该知道的

5.11.1 Visual Basic 的发展历程

Visual Basic 是由 BASIC 发展而来的，BASIC 语言已有数十年历史。BASIC 于 1964 年诞生于 Dartmouth 大学，它面对的是初级程序员。对已具备编程基础，准备深入功能更强大的编程语言的程序员来说，BASIC 通常是他们的第一门语言。

BASIC 是 Beginner's All-purpose Symbolic Instruction Code（初学者通用符号指令代码）的缩写，是国际上广泛使用的一种计算机高级语言。BASIC 简单、易学，目前，仍是计算机入门的主要学习语言之一。

BASIC 语言自问世经历了以下四个阶段：

第一阶段：（1964 年至 20 世纪 70 年代初）1964 年 BASIC 语言问世。

第二阶段：（1975 年至 20 世纪 80 年代中）微机上固化的 BASIC。

第三阶段：（20 世纪 80 年代中至 90 年代初）结构化 BASIC 语言。

第四阶段：（1991 年以来）Visual BASIC。

BASIC 是种易学易用的高级语言，非常适合初学者学习运用。常用的编译软件有 True BASIC，Turbo BASIC，Quick BASIC，Visual BASIC，CAREALIZER，GFA BASIC，POWER BASIC 等。

1991 年，微软推出了 Visual Basic 1.0 版。这在当时引起了很大的轰动。许多专家把 Visual Basic 的出现当作是软件开发史上的一个具有划时代意义的事件。其实，以我们现在的目光来看，Visual Basic 1.0 的功能实在是太弱了。但在当时，它是第一个“可视”的编程软件。这使得程序员欣喜之极，都尝试在 Visual Basic 的平台上进行软件创作。微软也不失时机地在四年内接连推出 Visual Basic 2.0、Visual Basic 3.0 及 Visual Basic 4.0 三个版本。并且从 Visual Basic 3.0 开始，微软将 ACCESS 的数据库驱动集成到了 Visual Basic 中，这使得 Visual Basic 的数据库编程能力大大提高。从 Visual Basic 4.0 开始，Visual Basic 也引入了面向对象的程序设计思想。Visual Basic 功能强大，学习简单。而且，Visual Basic 还引入了“控件”的概念，使得大量已经编好的 Visual Basic 程序可以被我们直接拿来使用，如今，Visual Basic 已经发展到了 Visual Basic 2010。

通过几年的发展，它已成为一种真正专业化的开发语言和环境。用户认为可用 Visual Basic 快速创建 Windows 程序，现在还可以编写企业水平的客户/服务器程序及强大的数据库应用程序，而它的最突出特性是快速创建一个单一的应用程序。这样一来，程序员们就可以花更多的时间来开发应用程序的功能，而不是在那些初级的、重复性的任务上浪费时间。Visual Basic 通常以“快速开发工具”而著称。

5.11.2 Visual Basic 2008 的解决方案

在前面的例子中间，可以看到每当建立了一个新的项目时，Visual Studio 自动将它加入到一个解决方案中。什么是解决方案？解决方案代表了 Visual Studio 的工作空间。而项目是 Visual Basic 的一个基本的工作单元，是一个应用程序相关组件（如程序代码、当前屏

幕的设置等）的管理器。解决方案则是一个或者多个项目的容器。在已经打开一个项目后直接再创建一个新的项目，此时有以下两种与解决方案相关的选项可以使用：

- **添入解决方案**。新建的项目将加入到目前打开的解决方案中。此时，解决方案将包含两个项目。
- **关闭解决方案**。关闭目前正在使用的任何解决方案，在新的解决方案中创建一个新的项目。这是默认的选项。

Visual Studio 也同时提供了解决方案资源管理器来组织和管理一个解决方案。在 Visual Studio 中同时最多只能打开一个解决方案。解决方案资源管理器如图 5-18 所示。

图 5-18　Visual Studio 2008 的解决方案资源管理器

在目录的最顶层是解决方案的名字，默认情况下和项目名称是一样的。可以在创建新项目时，在创建新项目的对话框上单击“更多”按钮来改变这一点。可以直接使用解决方案资源管理器来给项目添加新的项、直接添加新的项目、向项目中添加新的引用或者利用解决方案资源管理器进行定位。但需要注意的是，一个解决方案中只能包含一个起作用的 Sub Main 过程。

当新建一个项目时，Visual Basic 在指定的位置上创建该项目。这个位置默认时是 My Documents\Visual Studio 2008\Projects，创建完新的项目后，Visual Basic 已经在这个位置上创建了一些文件和文件夹，以程序例 5-17 为例，分别有以下几个文件和文件夹：

（1）创建了 Diving 文件夹，包含有所有解决方案要用到的文件。如果需要换台机器继续开发，则最简单是将 Diving 文件夹复制到其他机器上，然后打开 Diving 文件夹里的 Diving.sln 文件即可。

（2）在 Diving 文件夹里还包含有一个 Diving 文件夹。里面最主要的一个文件是 Module.vb。我们键入的 Visual Basic 代码就保存在这个文件里。My Project 文件夹里的文件和 Diving.vbproj 文件则是对本项目的一些配置的记录和 Visual Studio 2008 工作环境的一些配置文件。

（3）另外还有两个文件夹 bin 和 obj 用来存放编译后的输出结果。如果对项目进行了编译，则还会有其他的中间文件和用户配置文件产生。特别注意 bin 文件夹里的 Debug 文件夹内有 Diving.exe 文件。当程序正确编译后会有这个文件，该文件可以在装 Windows 操作系统的机器上直接运行（XP 下需要安装.NET）而无须安装 Visual Studio。

5.11.3 良好的编程风格

为了做好程序设计，必须首先分析所给问题，明确要求。标识输入量与输出量，确定它们的数据类型，然后确定从所给输入到输出需执行的步骤，即进行算法设计。算法设计应自顶向下，逐步求精。最后按照算法写出相应的程序。在编写程序时应正确使用 Visual Basic 语句，并要特别注意正确使用标点符号，不要漏用或错用。以下是一些建议：

- 在程序中最好每行只包括一个语句。
- 每次只声明一个变量。
- 变量和控件名尽可能起的有意义，容易记忆和理解并且风格始终保持一致。
- 注意使用语句的缩格与按层次对齐，采用锯齿状的书写方式。
- 在必要的地方加上注释，以提高程序的可读性。
- 要编写出易读的代码，另一个重要的方面是留出大量的空白。以告诉读者每一段代码都是一个工作单元。
- 模仿本书的例子或其他好的编程风格的例子。

习题

1．给出 3 个整数，求它们的和与平均值。

2．自来水公司采取按用水量阶梯式计价的办法，居民应交水费 y（元）与月用水量 x（吨）相关，函数关系式如下。编写程序计算当 x1=12，x2=30 时 y 的值。

$$y = f(x) = \begin{cases} 0 & (x<=0) \\ 4x/3 & (0<x<=15) \\ 2.5x-10.5 & (x>15) \end{cases}$$

3．设 x1=0133 表示火车 1 点 33 分开出，x2=2209 表示火车 22 点 09 分到站。x1 和 x2 都是整数，计算火车运行的时间 y（运行时间假设小于 24h），同样用一个 4 位整数表示，前 2 位为小时数，后两位是分钟数。

4．计算序列 2/1+3/2+5/3+8/5+…的前 n 项之和。

5．给定一个含有 10 个整数的数组，判断 x 是否在数组中。如是，将 x 在数组中的位置（下标）存于变量 y 中，否则 y 的值为–1。

6．将第 2 题用过程调用的方式实现。例如：

```
Function WaterFee (ByVal x As Deciaml) As Decimal
```

7．将第 5 题用过程的方式实现。例如：

```
Function IsHere (ByVal A() As Integer, ByVal x As Integer) As Integer
```

8．某公司员工的工资计算方法如下，一周内工作时间 40h 之内（含 40h），按正常工作时间计酬，超出 40h 的工作时间部分，按正常工作时间报酬的 1.5 倍计酬。员工按进公司时间分为新职工和老职工，新职工的正常工资为 30/h，老职工的正常工资为 50/h。进公司 5 年以上（含 5 年）的员工为老职工，5 年以下的为新职工）， 请按该计酬方式计算员工的工资。要求输入员工进公司的一周工作时间、工作年数，输出其一周的工资，保留 2 位小数。

9．输入年份、月份、日子，输出这一天是该年中的第几天。如输入 3 个整数，2009 3 2，则输出 This is the 61th of 2009。

10．一只猴子第一天摘下若干个桃子，当即吃了一半，还不过瘾，又多吃了一个；第二天早上又将剩下的桃子吃掉一半，又多吃了一个。以后每天早上都吃了前一天剩下的一半零一个。到第 n 天（1＜n＜=10，从键盘输入）早上想再吃时，见只剩下一个桃子了。问第一天共摘了多少个桃子？

11．总共 50 件商品，有两种构成，钥匙扣 2 元一个，漫画书 4 元一本，要卖出 160 元，应如何搭配（输出所有可能的配对情况）？

12．打印输出所有“水仙花数”。“水仙花数”是指一个三位的正整数，其各位数字立方和等于该数本身。例如：153 是一个“水仙花数”，因为 $153=1^3+5^3+3^3$。

13．编写过程 IsSquare，判断某个自然数是否为平方数。是则返回 True，不是则返回 False。

14．求数组中出现次数最多的数及出现次数。数组为整数，8 个数。输出出现最多的数及次数。

15．一个自然数是素数，且它的各位数字位置经过任意对换后仍为素数，则称为绝对素数。例如，13 是绝对素数。输出所有两位数的绝对素数。

第 6 章　数据结构

引言

我们已经知道用计算机解决一个具体问题时，首先要从具体问题抽象出一个适当的数学模型，然后设计一个解此数学模型的算法，最后编出程序，进行测试、调整直至得到最终解答。如果一个问题可以用数学的方程来描述，如人口增长可以用微分方程来描述，只需要输入相应的数据，然后通过计算机求解该方程即可，这一般称为数值计算。然而，很多问题是无法用数学方程来描述的，如计算机和人的对弈问题。这类非数值计算问题需要用其他的方式来描述和求解，其核心是数据结构和算法。

本节对常用的数据结构作了简单讲述，以及对常用的查找和排序算法也作了简单的讲述。本节同时还就.NET 对常用数据结构提供的支持类作了简单的讲述。

教学目的

- 掌握数据与数据结构的基本概念。
- 掌握线性表的基本结构及运算。
- 掌握栈和队列的基本概念。
- 了解树和图的基本概念。
- 掌握查找和排序的初步技术。

6.1　数据与数据结构

6.1.1　数据

什么是数据？数据是描述客观事物的信息符号的集合，这些信息符号能被输入到计算机中存储起来，又能被程序处理、输出。事实上，数据本身是随着计算机的发展而不断扩展的概念。在计算机发展的初期由于计算机主要用于数值计算，数据指的就是整数、实数等数值；在计算机用于文字处理时，数据指的就是由英文字母和汉字组成的字符串；随着计算机硬件和软件技术的不断发展，扩大了计算机的应用领域，如表格、图形、图像和声音等也属于数据的范畴。目前，非数值问题的处理占用的 90%以上计算机时间。

数据类型是程序设计中的概念，程序中的数据都属于某个特殊的数据类型，它是指具有相同特性的数据的集合。数据类型决定了数据的性质，如取值范围、操作运算等。常用的数据类型有整型、浮点型、字符型等。数据类型还决定了数据在内存中所占空间的大小，如字符型占一个字节，而长整型一般占 4B 等。

对于复杂一些的数据，仅用数据类型无法完整地描述，如表示教师得分要描述教师的姓名、各项得分，这时需要用到数据元素的概念。数据元素中可能用到多个数据类型（称为数据项），共同描述一个客体，如教师。数据元素有时又称为记录或结点。在程序设计中，前面所说的数据类型又称为基本数据类型，由基本数据类型组成的数据元素的定义称为构造数据类型（结构和类都属此列）。

教师得分登记表

姓　名	教学得分	科研得分	其他得分	合计
张　力	35	34	11	80
王　五	36	35	12	83
…	…	…	…	…

教师得分登记表的数据元素是姓名、教学得分、科研得分、其他得分、合计，也就是说，每个数据元素由姓名、教学得分、科研得分、其他得分、合计五个数据项组成。这五个数据项含义明确，若再细分就无明确独立的含义，属于基本数据类型（字符型、整型或浮点型）。

6.1.2 数据结构

计算机的处理效率与数据的组织形式和存储结构密切相关。这类似于人们所用的《英语词典》、《科学技术辞海》和《新华字典》等工具书，它们都是按字母或拼音字母的顺序组织排列“词条”。这样人们查阅工具书的速度较快。假如“词条”不是按字母顺序组织排列，而是任意顺序组织排列，那么查词速度一定很低。因此，很有必要研究数据的组织形式和存储结构。另外在当今网络世界中传递数据更加依赖于数据的组织形式和存储结构。

什么是数据结构？数据结构在计算机科学界至今没有标准的定义。个人根据各自的理解的不同而有不同的表述方法。

Sartaj Sahni 在他的《数据结构、算法与应用》一书中称：“数据结构是数据对象，以及存在于该对象的实例和组成实例的数据元素之间的各种联系。这些联系可以通过定义相关的函数来给出。”他将数据对象（Data Object）定义为“一个数据对象是实例或值的集合”。

Clifford A • Shaffer 在《数据结构与算法分析》一书中的定义：“数据结构是 ADT（Abstract Data Type，抽象数据类型）的物理实现。”

Lobert L • Kruse 在《数据结构与程序设计》一书中，将一个数据结构的设计过程分成抽象层、数据结构层和实现层。其中，抽象层是指抽象数据类型层，它讨论数据的逻辑结构及其运算，数据结构层和实现层讨论一个数据结构的表示和在计算机内的存储细节及运算的实现。

由此可见，在任何问题中，构成数据的数据元素并不是孤立存在的，它们之间存在着一定的关系以表达不同的事物及事物之间的联系。所以，简单地说，数据结构就是研究数据及数据元素之间关系的一门学科，它包括以下三个方面的内容：

- 数据的逻辑结构。
- 数据的存储结构。
- 数据的运算（即数据的处理操作）。

一般认为，一个数据结构是由数据元素依据某种逻辑联系组织起来的。对数据元素间

逻辑关系的描述称为数据的逻辑结构；数据必须在计算机内存储，数据的存储结构是数据结构的实现形式，是其在计算机内的表示；此外，讨论一个数据结构必须同时讨论在该类数据上执行的运算才有意义。

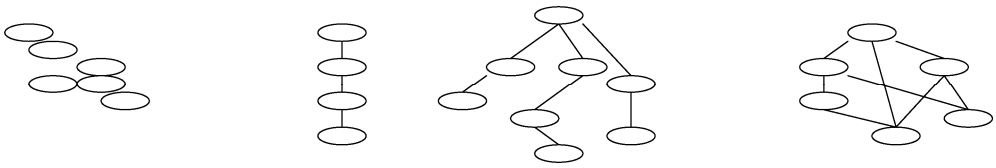
1）*数据的逻辑结构*

数据的逻辑结构就是数据元素之间的逻辑关系。这里，我们对数据所描述的客观事物本身的属性意义不感兴趣，只关心它们的结构及关系。将那些在结构形式上相同的数据抽象成某一数据结构，如线性表、树和图等。

根据数据元素之间关系的不同特性，数据结构又可分为以下四大类（图 6-1）：

- **集合**。数据元素之间的关系只有“是否属于同一个集合”。
- **线性结构**。数据元素之间存在线性关系，即最多只有一个前导和一个后继元素。
- **树形结构**。数据元素之间呈层次关系，即最多有一个前导和多个后继元素。
- **图状结构**。数据元素之间的关系为多对多的关系。

其中，树和图又统称为非线性数据结构。



（a）集合　（b）线性结构　（c）树形结构　（d）图状结构

图 6-1　四种逻辑结构示意图

2）*数据的存储结构*

数据的逻辑结构是从逻辑上来描述数据元素之间的关系的，是独立于计算机的。然而讨论数据结构的目的是为了在计算机中实现对它的处理。因此，还需要研究数据元素和数据元素之间的关系如何在计算机中表示，这就是数据的存储结构，又称为数据的映像。

计算机的存储器是由很多个存储单位组成的，每个存储单元有唯一的地址。数据的存储结构要讨论的就是数据结构在计算机存储器上的存储映像方法。根据数据结构的形式定义，数据结构在存储器上的映像，不仅包括数据元素集合如何存储映像，而且还包括数据元素之间的关系如何存储映像。

一般来说，数据在存储器中的存储有四种基本的映像方法。

（1）顺序存储结构。就是把数据元素按某种顺序放在一块连续的存储单元中，其特点是借助数据元素在存储器中的相对位置来表示数据元素之间的关系。顺序存储的问题是，如果元素集合很大，则可能找不到一块很大的连续的空间来存放。

（2）链式存储结构。有时往往存在这样一些情况：一种情况是存储器中没有足够大的连续可用空间，只有不相邻的零碎小块存储单元；另一种情况是在事前申请一段连续空间时，因无法预计所需存储空间的大小，需要临时增加空间。所有这些情况表明，要得到一块合适的连续存储单元并非易事，即这种情况下顺序存储结构无法实现。

链式存储结构的特点就是将存放每个数据元素的结点分为两部分：一部分存放数据元素（称为数据域）；另一部分存放指示存储地址的指针（称为指针域），借助指针表示数据元素之间的关系。结点的结构如下：

数据域	指针域

链式存储结构可用一组任意的存储单元来存储数据元素，这组存储单元可以是连续的，也可以是不连续的。链式存储因为有指针域，增加了额外的存储开销，并且实现上也较为麻烦，但大大增加了数据结构的灵活性。

（3）索引存储结构。在线性表中，数据元素可以排成一个序列：R_1，R_2，R_3，…，R_n，每个数据元素 R_i 在序列里都有对应的位置码 i，这就是元素的索引号。索引存储结构就是通过数据元素的索引号 i 来确定数据元素 R_i 的存储地址。一般索引存储结构有两种实现方法：① 建立附加的索引表，索引表里第 i 项的值就是第 i 个元素的存储地址；② 当每个元素所占单元数都相等时，可用位置码 i 的线性函数值来确定元素对应的存储地址，即

$$Loc(R_i)=(i-1)*L+d_0$$

（4）散列存储结构。这种存储方法就是在数据元素与其在存储器上的存储位置之间建立一个映像关系 F。根据这个映像关系 F，已知某数据元素就可以得到它的存储地址。即 D=F（E），这里 E 是要存放的数据元素，D 是该数据元素的存储位置。可见，这种存储结构的关键是设计这个函数 F，但函数 F 不可能解决数据存储中所有问题，还应有一套意外事件的处理方法，它们共同实现数据的散列存储结构。哈希表就是一种常见的散列存储结构。

3）数据的运算

数据的运算是定义在数据逻辑结构上的操作，如插入、删除、查找、排序和遍历等。每种数据结构都有一个运算的集合。

6.2 线性表

线性表是最基本、最简单也是最常用的一种数据结构。线性表中数据元素之间的关系是一对一的关系，即除了第一个和最后一个数据元素之外，其他数据元素都是首尾相接的。线性表的逻辑结构简单，便于实现和操作，在实际应用中是广泛采用的一种数据结构。

6.2.1 线性表的逻辑结构及运算

线性表是一个线性结构，它是一个含有 n≥0 个结点的有限序列，对于其中的结点，有且仅有一个开始结点（第一个结点）并且没有前驱结点但有一个后继结点，有且仅有一个终端结点（最后一个结点）并且没有后继结点但有一个前驱结点，其他的结点都有且仅有一个前驱结点和一个后继结点。

一般地，一个线性表可以表示成一个线性序列：k1,k2,…,kn，其中 k1 是开始结点，kn 是终端结点。线性表具有以下一些基本性质：

- 数据元素的个数 n 定义为表的长度，当 n=0 时，称为空表。空表中无数据元素。
- 若表非空，则必存在唯一的一个开始结点。
- 必存在唯一的一个终端结点。
- 除最后一个元素之外，其余结点均有唯一的后继结点。
- 除第一个元素之外，其余结点均有唯一的前驱结点。
- 数据元素 ki（1≤i≤n）在不同情况下的具体含义不同，它可以是一个数，或者是一个符号，或者是更复杂的信息。虽然不同数据表的数据元素可以是各种各样的，但对于同

一线性表的各数据元素必定具有相同的数据类型和长度。

例 6-1 线性表的例子。

（1）某班学生的数学成绩（78，92，66，84，45，72，92）是一个线性表，每个数据元素是一个正整数，表长为 7。

（2）一星期的七天的英文缩写词（SUN，MON，TUE，WED，THU，FRI，SAT）是一线性表，表中数据元素是一字符串，表长为 7。

（3）某企业职工基本工资情况（(张三，助工，3 543)，(李四，高工，21 986)，(王五，工程师，9731)）也是一线性表，表中数据元素是由姓名、职称、工龄、基本工资四个数据项组成的一个记录（对象），表长为 3。

线性表可以进行的常用基本操作有以下几种：

- **置空表**。将线性表 L 的表长置为 0。
- return。求出线性表 L 中数据元素的个数。
- **取表中元素**。仅当 1≤i≤Length(L)时，取得线性表 L 中的第 i 个元素 ki（或 ki 的存储位置），否则无意义。
- **取元素 ki 的直接前趋**。当 2≤i≤Length(L)时，返回 ki 的直接前趋 ki–1。
- **取元素 ki 的直接后继**。当 1≤i≤Length(L)–1 时，返回 ki 的直接后继 ki+1。
- **定位**。返回元素 x 在线性表 L 中的位置。若在 L 中有多个 x，则只返回第一个 x 的位置，若在 L 中不存在 x，则返回 0。
- **插入**。在线性表 L 的第 i 个位置上插入元素 x，运算结果使得线性表的长度增加 1。
- **删除**。删除线性表 L 的第 i 个位置上的元素 ki，此运算的前提应是 Length(L)≠0，运算结果使得线性表的长度减 1。

对线性表还有一些更为复杂的操作，例如，将两个线性表合并成一个线性表；将一个线性表分解为 n 个线性表；对线性表中的元素按值的大小重新排列等。这些运算都可以通过上述 8 种基本运算的组合派生来实现。

6.2.2 线性表的存储结构

要使线性表成为计算机可以处理的对象，就必须把线性表的数据元素及数据元素之间的逻辑关系都存储到计算机的存储器中。线性表常用顺序方式和链表方式来存储。

1．线性表的顺序存储

线性表的顺序存储结构就是将线性表的每个数据元素按其逻辑次序依次存放在一组地址连续的存储单元里。由于逻辑上相邻的元素存放在内存的相邻单元中，所以线性表的逻辑关系蕴含在存储单元的物理位置相邻的关系中。也就是说，在顺序存储结构中，线性表的逻辑关系的存储是隐含的。

设线性表中每个元素占用 C 个存储单元，用 Loc(ki)表示元素 ki 的存储位置，则顺序存储结构的存储示意图如图 6-2 所示。

从图 6-2 中可以看出，若已知线性表的第一个元素的存储位置是 Loc(k1)，则第 i 个元素的存储位置为

$$loc(ki)=loc(k1)+c*(i-1) \qquad 1\leqslant i\leqslant n$$

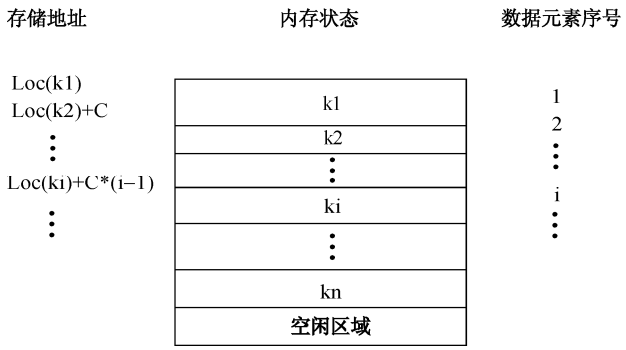


图 6-2　线性表的顺序存储结构示意图

可见，线性表中每个元素的存储地址是该元素在表中序号的线性函数。只要知道某元素在线性表中的序号就可以确定其在内存中的存储位置。所以，线性表的顺序存储结构是一种随机存取结构。

在 Visual Basic 语言中，数组是在内存中连续分配的。所以数组天生就是一种线性结构。用数组来实现线性表，可以预先定义一个较大的数组，用来存放线性表中的元素。元素从数组的 0 位置存起，数组最后的一些位置是空闲的。

例 6-2　一个整数线性表的实现。用整型数组存储元素，实现线性表的基本操作。

分析：使用数组 list 来存储元素。list 的大小设为 MAX=1000。表长用 n 来表示，线性表中的每一个元素都是整数。这里，使用一个自定义的数据类型来描述线性表。该数据类型 ListType 包含一个数组用于存放数据。一个整数表示表长。

程序：

```
Module Module1
    Structure ListType
        Dim Data() As Integer
        Dim n As Integer
    End Structure

    Sub main()
        Const MAX As Integer = 1000
        Dim list As ListType
        ReDim list.Data(MAX)

        '以下是对线性表的测试

        Initiate(list)   '置表为空

        '插入 10 个元素并显示
        For i As Integer = 0 To 9
            Insert(list, i, i + 1)
        Next
        Print(list)

        '在位置 5 插入 99 并显示
```

```
        Insert(list, 5, 99)
        Print(list)

        '删除第 8 个元素
        Delete(list, 8)
        Print(list)

        '显示第 3 个元素的前驱
        Console.WriteLine("第 3 个元素的前驱是{0}", PriorData(list, 3))

    End Sub

    '置空表
    Sub Initiate(ByRef L As ListType)
        L.n = 0    '将表的长度置为 0
    End Sub

    '求表长
    Function GetLength(ByVal L As ListType) As Integer
        Return L.n
    End Function

    '取表中元素
    Function GetData(ByVal L As ListType, ByVal i As Integer) As Integer
        If i >= 0 And i < L.n Then
            Return L.Data(i)
        Else
            Console.WriteLine("要求的元素不存在，程序终止")
            End
        End If
    End Function

    '取元素 ki 的直接前趋
    Function PriorData(ByVal L As ListType, ByVal i As Integer) As Integer
        If i >= 1 And i < L.n Then
            Return L.Data(i - 1)
        Else
            Console.WriteLine("要求的元素不存在，程序终止")
            End
        End If
    End Function

    '取元素 ki 的直接后继
    Function NextData(ByVal L As ListType, ByVal i As Integer) As Integer
        If i >= 0 And i < L.n - 1 Then
            Return L.Data(i + 1)
        Else
```

```
            Console.WriteLine("要求的元素不存在，程序终止")
            End
        End If
    End Function

'定位：返回元素 x 在线性表 L 中的位置。
'若在 L 中有多个 x，则只返回第一个 x 的位置，若在 L 中不存在 x，则返回-1。
    Function Locate(ByVal L As ListType, ByVal x As Integer) As Integer
        Dim i As Integer = 0
        While i < L.n
            If x = L.Data(i) Then
                Return i
            End If
            i = i + 1
        End While
        Return -1
    End Function

    '插入：在线性表 L 的第 i 个位置上插入元素 x，运算结果使得线性表的长度增加 1
    Sub Insert(ByRef L As ListType, ByVal i As Integer, ByVal x As Integer)
        If i >= 0 And i <= L.n Then
            '将 i 以后的元素向后移动一个位置
            Dim j As Integer = L.n        '最后一个元素的后一个位置
            While j > i
                L.Data(j) = L.Data(j - 1)
                j = j - 1
            End While
            L.Data(i) = x    '插入 x
            L.n = L.n + 1    '表长增加 1
        Else
            Console.WriteLine("插入的位置不正确，程序终止")
            End
        End If
    End Sub

    '删除线性表 L 的第 i 个位置上的元素 ki
    Sub Delete(ByRef L As ListType, ByVal i As Integer)
        If i >= 0 And i < L.n Then
            '将 i+1 以后的元素依次前移一个位置
            While i < L.n
                L.Data(i) = L.Data(i + 1)
                i = i + 1
            End While
            L.n = L.n - 1
        Else
            Console.WriteLine("删除的位置不正确，程序终止")
            End
```

```
            End If
        End Sub

        '打印，将表中的元素全部依次打印出来，为了演示方便
        Sub Print(ByVal L As ListType)
            For i As Integer = 0 To L.n - 1
                Console.Write(L.Data(i).ToString() + " ")
            Next
            Console.WriteLine("表长为：{0}", L.n)
        End Sub
    End Module
```

需要注意的是，由于 Visual Basic 中的数组下标是从 0 开始的。方便起见，本程序实现的线性表默认的第一个元素下标是 0。因此，在第 5 个位置上插入，实际是在线性表的第 6 个位置上插入。也可以修改程序，使其和前面描述的线性表一致。还要注意的是，ByVal 和 ByRef 的使用，一般的，要改变 list 中的值（尤其是 n 的值）应该使用 ByRef。图 6-3 所示为程序运行的结果。

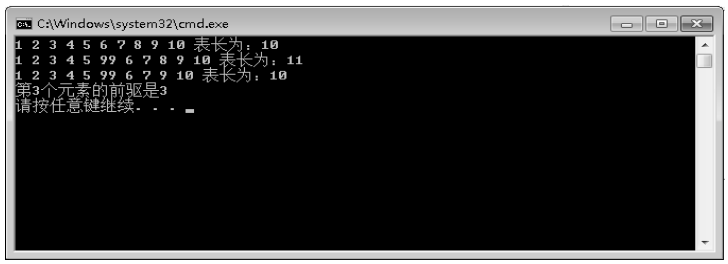


图 6-3　例 6-2 的线性表程序运行的结果

2．线性表的单链表存储

线性表的顺序存储结构是把整个线性表存放在一片连续的存储区，其逻辑关系上相邻的两个元素在物理位置上也相邻，因此，可以随机存取表中任一元素，每个元素的存储位置可用一个简单、直观的公式来表示。然而，某一线性表中的元素频繁进行插入和删除操作时，为了保持元素在存储区的连续性，在插入元素时必须移动大量元素给新插入的元素“腾位置”，而在删除时，又必须移动大量后继元素“补缺”，因而在操作执行时要花大量时间去移动数据元素。

能否设计一种新的存储结构，在元素插入、删除时无须改变已存储元素的位置？这就是我们将讨论的另一种存储结构——链式存储结构。

用链式方式存储一个线性表，其特点是用一组任意的存储区存储该线性表，此存储区可以是连续的，也可以是分散的。这样，逻辑上相邻的元素在物理位置上就不一定是相邻的，为了能正确反映元素的逻辑顺序，就必须在存储每个元素 a_i 的同时，存储其直接后继（或直接前趋）的存储位置。

链式存储方式有很多种，我们在这里仅介绍单链表。在单链表中每个结点都由两部分组成：存储数据元素的数据域（Data）；存储直接后继结点存储位置的指针域（Next）。其结点结构如下：

Data	Next

一个由学生姓名组成的线性表（张三，李四，王五，赵六），采用单链表为存储结构时，其单链表的逻辑示意图如图 6-4 所示。

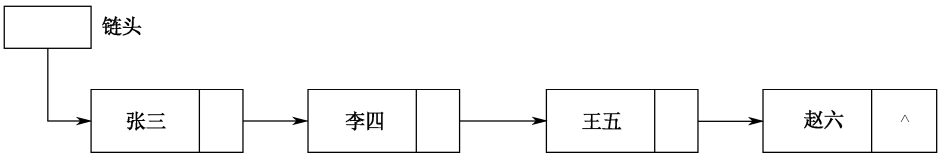


图 6-4　单链表的逻辑结构

在链式存储结构中，不可避免地要提到“指针”这个概念，在此读者可以简单地将其理解为某一存储单元的“位置”，在实际的语言实现中，有些语言本身有指针类型的变量（如 C 和 Pascal），有些语言如（Visual Baisc 和 Java）需要其他方式来实现。

在图 6-4 所示的单链表中，链头是指向单链表中第一个结点的指针，称为头指针；最后一个元素赵六所在结点不存在后继，因而其指针域为“空”（用 NULL 或∧表示）。该单链表在存储区的物理映像如图 6-5 所示。

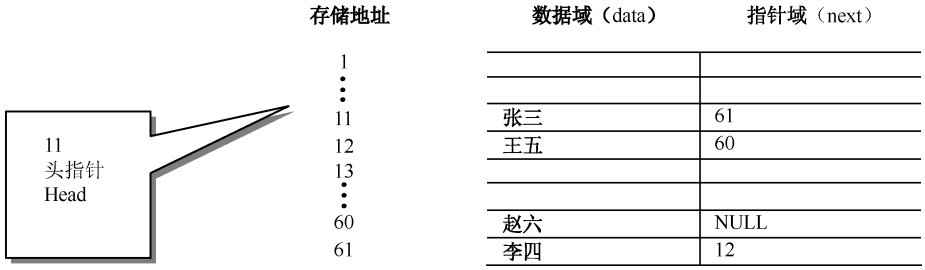


图 6-5　单链表在存储区的物理映像

在单链表存储结构下，顺序表的插入和删除这两个操作的实现方法如下所述。

1）单链表的插入

设有线性表（a_1，a_2，…，a_i，a_{i+1}，…，a_n），用单链表存储，头指针为 Head，要求在存储数据元素 a_i 的结点之前插入一个数据元素为 X 的结点。设新插入的结点指针是 S。

若已知 a_i 的前趋 a_{i-1} 所在结点的指针 P，只要执行以下两步操作即可：

步骤 1　令结点 S 的指针域指向 a_i 所在的结点（S→next=P→next）；

步骤 2　令结点 P 的指针域指向结点 S（P→next=S）。

执行插入后的单链表的逻辑状态如图 6-6 所示。

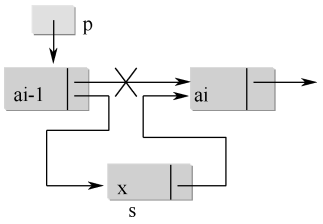


图 6-6　在带头结点的单链表中插入结点 S

由此可见，插入操作执行之前，首先就是要找到单链表中插入位置的前一个结点的指针（存储位置）。由于知道头指针，可以从头指针一一找到下一个元素，直到所需的元素。与顺序存储方式的随机访问相比，这是链式存储的不便之处。

2）单链表的删除

删除操作和插入操作一样，首先要搜索单链表以找到指定删除结点的前趋结点（假设为 p），然后只要将待删除结点的指针域内容赋予 p 结点的指针域就可以了。删除元素所在的结点之后，单链表的逻辑状态如图 6-7 所示。

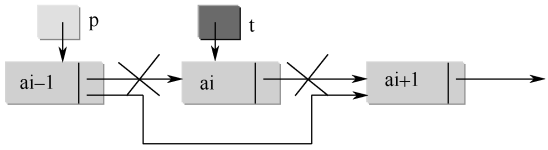


图 6-7　从带头结点的单链表中删除一个结点

6.2.3　List 类

在 Visual Basic 中，线性表除了使用数组自己编程实现外，还可以通过 Visual Basic 提供的 List 类来实现。按照使用类的方法，应该先声明一个 List 类的对象，这样就有了一个线性表对象。需要注意的是，List 类是一个泛型类，关于泛型是一个较复杂的概念。在此，简单的理解为创建一个 List 对象时需要指明将在 List 中存储什么类型的数据。如声明一个整数类型的 List 对象的格式如下：

```
Dim listA As New List(of Integer)
```

例 6-3　使用 List 类实现例 6-2。

程序：

```
Module Module1

    Sub Main()
        Dim listA As New List(Of Integer)

        '插入 10 个元素并显示
        For i As Integer = 0 To 9
            listA.Insert(i, i + 1)
        Next
        Print(listA)

        '在位置 5 插入 99 并显示
        listA.Insert(5, 99)
        Print(listA)

        '删除第 8 个元素
        listA.RemoveAt(8)
        Print(listA)

        '排序
        listA.Sort()
        Print(listA)

        '反转
        listA.Reverse()
        Print(listA)
    End Sub

    Sub Print(ByVal L As List(Of Integer))
```

```
        For i As Integer = 0 To L.Count - 1
            Console.Write(L(i).ToString() + " ")
        Next
        Console.WriteLine("表长为：{0}", L.Count)
    End Sub

End Module
```

程序的运行结果如图 6-8 所示。

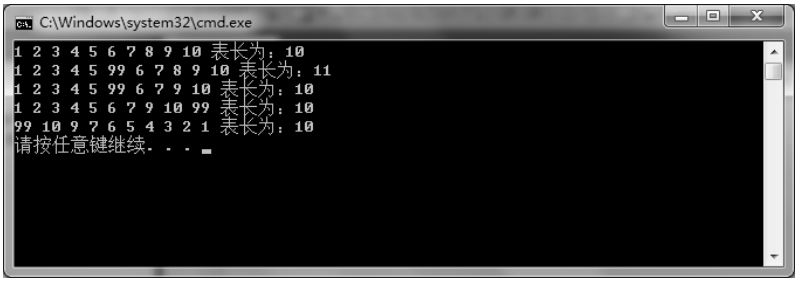


图 6-8　使用 List 类实现的线性表

可以看到程序的运行结果和例 6-2 基本相似，但有一些区别。List 类并没有提供前驱或者后继方法，但这通过下表的操作很容易实现。其次，List 类提供了排序方法，简单的调用该方法可以对 List 中的元素排序。但是需要注意的是，在排序时，程序要知道如何比较 List 中两个元素的大小。由于可以将自定义的数据类型存储到 list 中，此时程序就不清楚如何比较大小了，因而也无法排序（或者说排序需要自己提供一个比较的方法，这已超出了本书的讨论范围）。最后，还可以看到由于使用了 Visual Basic 提供的 List，例 6-3 程序要比例 6-2 的短小得多而功能却更强大。对 List 类中常用的方法如下：

- Add。将对象添加到 List 的结尾处。
- Clear。从 List 中移除所有元素（置空表）。
- Contains。确定某元素是否在 List 中。如在，Contains 返回 True，否则返回 False。
- FindIndex。搜索与指定条件相匹配的元素，返回 List 或它的一部分中第一个匹配项的从零开始的索引。
- IndexOf。返回 List 或它的一部分中某个值的第一个匹配项的从零开始的索引。
- Insert。将元素插入 List 的指定索引处。
- LastIndexOf。返回 List 或它的一部分中某个值的最后一个匹配项的从零开始的索引。
- Remove。从 List 中移除特定对象的第一个匹配项。
- RemoveAt。移除 List 的指定索引处的元素。
- Reverse。将 List 或它的一部分中元素的顺序反转。
- Sort。对 List 或它的一部分中的元素进行排序。

List 中最常用的属性只有一个：List.Count 表示 List 表中的元素个数（表长）。

6.2.4　LinkedList 类

和 List 类一样，Visual Basic 提供了 LinkedList 类用于帮助实现链表。LinkedList 类通常还需要和 LinkedListNode 类一起使用，LinkedListNode 类用于表示链表中的一个结点。

在本节中只以一个简单的例子来说明一下，读者要是有兴趣请在微软的网站（MSDN）上搜索进一步的资料。

例 6-4 对图 6-4 所示的链表做一个简单的实现。

分析：图 6-4 的链表中每一个结点存储的数据类型是一个字符串。先生成一个空的链表，然后用给的名字创建每一个结点，并加入到链表中。程序还编写了一个 Display 过程用于显示链表的内容。

程序：

```
Module Module1

    Sub Main()
        '定义一个空的链表
        Dim linkedListA As New LinkedList(Of String)

         '生成链表的 4 个结点
        Dim linkedListNodeA As New LinkedListNode(Of String)("张三")
        Dim linkedListNodeB As New LinkedListNode(Of String)("李四")
        Dim linkedListNodeC As New LinkedListNode(Of String)("王五")
        Dim linkedListNodeD As New LinkedListNode(Of String)("赵六")

         '将结点加入到链表中，实际运用中不一定要依照下面的顺序加入
         '在此，为了演示 AddFirst、AddLast、AddAfter 和 AddBefore 的用法
         '而这么做的。在首尾加入链表时（调用 AddFirst 和 AddLast），
         '也可以不用声明 LinkedListNode 对象，而直接加入，如：
         ' linkedListA.AddFirst("张三")

        linkedListA.AddFirst(linkedListNodeA)
        linkedListA.AddLast(linkedListNodeD)
        linkedListA.AddAfter(linkedListNodeA, linkedListNodeB)
        linkedListA.AddBefore(linkedListNodeD, linkedListNodeC)
        DisPlay(linkedListA)

         '移除第一个结点
        linkedListA.RemoveFirst()
        DisPlay(linkedListA)

         '移除结点 C
        linkedListA.Remove(linkedListNodeC)
        DisPlay(linkedListA)

    End Sub

    Sub Display(ByVal LL As LinkedList(Of String))
        For Each name As String In LL
            Console.Write(name + " ")
        Next
        Console.WriteLine("表长为：{0}", LL.Count)
```

```
    End Sub

End Module
```

6.3 栈和队列

栈和队列也是线性结构，线性表、栈和队列这三种数据结构的数据元素，以及数据元素间的逻辑关系完全相同，差别是线性表的操作不受限制，而栈和队列的操作受到限制。栈的操作只能在表的一端进行，队列的插入操作在表的一端进行而其他操作在表的另一端进行，所以，把栈和队列称为操作受限的线性表。

6.3.1 栈

栈是只能在某一端插入和删除的特殊线性表。它按照后进先出的原则存储数据，先进入的数据被压入栈底，最后的数据在栈顶，需要读数据的时候从栈顶开始弹出数据（最后一个数据被第一个读出来）。

栈是允许在同一端进行插入和删除操作的特殊线性表。允许进行插入和删除操作的一端称为栈顶（top），另一端为栈底（bottom）；栈底固定，而栈顶浮动；栈中元素个数为零时称为空栈。插入一般称为进栈（PUSH），删除则称为出栈（POP）。栈又称为先进后出表。

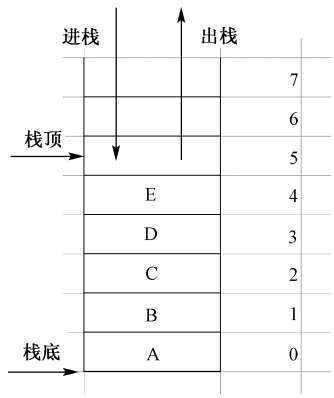


图 6-9　栈的示意图

图 6-9 是一个栈的示意图。

栈顶始终指向栈顶最后一个元素之后的空位置。在图 6-9 中，栈里面共有 5 个元素，入栈的次序依次是 A B C D E。栈底始终等于 0，而栈顶等于 5。图 6-10 则描述了最后两个元素出栈。F 进栈的情形。图 6-10（a）最后一个元素 E 出栈，栈顶=4；（b）D 出栈，栈顶=3；（c）元素 F 进栈，栈顶=4。

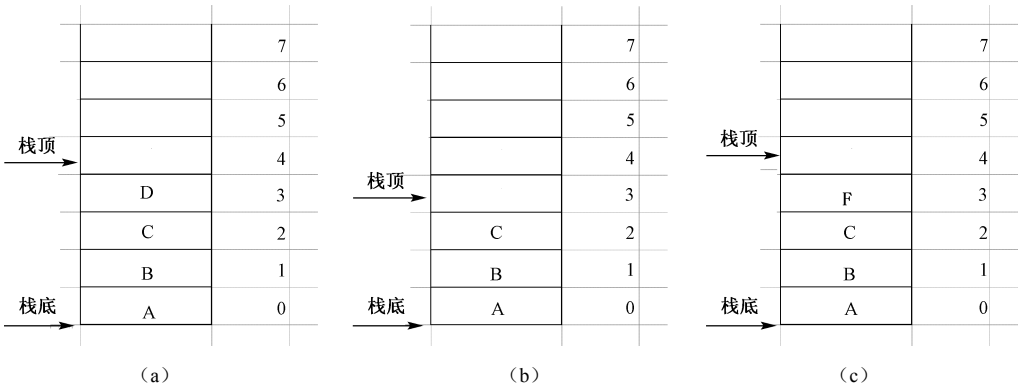


图 6-10　最后两个元素出栈、卡进栈的情形

当栈中没有元素时，称为空栈，空栈的条件是栈顶=栈底。栈的大小一般是预先定义好的，当栈顶=栈的大小时，称为栈满。很显然的，当栈为空时，不能进行出栈操作，而当栈

满时不能进行入栈操作。一般的，对栈有如下几个操作：

- **求栈的长度**。GetLength，返回栈中数据元素的个数。
- **判断栈是否为空**。IsEmpty，如果栈为空返回 True，否则返回 False。
- **清空栈**。Clear，使栈为空。
- **入栈操作**。Push 将新的数据元素添加到栈顶，栈发生变化。
- **出栈操作**。Pop 将栈顶元素从栈中取出，栈发生变化。
- **取栈顶元素**。GetTop 返回栈顶元素的值，栈不发生变化。

同样的，栈在计算机中的存储结构，也有顺序存储和链式存储结构。显然顺序结构自然可以用一个数组来实现。

例 6-5 用数组实现栈。

分析：用一个指定大小的数组来存储栈的内容。在此，假设栈里存储的是字符串。变量 Top 保存栈顶的数组下标，变量 Bottom 为栈底，始终为 0。

程序：

```
Module Module1
    '定义栈
    Structure Stack
        Dim sArray() As String
        Dim top As Integer
        Dim bottom As Integer
    End Structure

    Sub Main()
        '声明一个栈
        Dim stackExamp As Stack
        '栈空间大小为 30
        ReDim stackExamp.sArray(30)

        '初始栈，置为空栈
        Clear(stackExamp)

        '将一些字符串压入栈
        Push(stackExamp, "西安")
        Push(stackExamp, "交通")
        Push(stackExamp, "大学")

        '只要栈不空，将栈里的字符串全部弹出栈并显示
        While Not IsStackEmpty(stackExamp)
            Console.Write(Pop(stackExamp) + " ")
        End While

    End Sub

    '清空栈，也就是将栈顶和栈底指针设为 0
    Sub Clear(ByRef s As Stack)
        s.top = 0
```

```
        s.bottom = 0
    End Sub

    '入栈，存入数组中，同时栈顶加一
    '注意此处有不太完善的地方，也就是没有检测栈是否满了
    Sub Push(ByRef s As Stack, ByVal item As String)
        s.sArray(s.top) = item
        s.top = s.top + 1
    End Sub

    '检查栈是否为空
    Function IsStackEmpty(ByVal s As Stack) As Boolean
        If s.top = s.bottom Then
            Return True
        Else
            Return False
        End If
    End Function

    '出栈函数，检查了栈里是否有元素
    Function Pop(ByRef s As Stack) As String
        If s.top > s.bottom Then
            s.top = s.top - 1
            Return s.sArray(s.top)
        Else
            Return vbNull   '如果栈空，则返回一个空值
        End If
    End Function

    '得到栈中元素的个数
    Function GetLength(ByVal s As Stack) As Integer
        Return s.top - s.bottom
    End Function

End Module
```

程序的运行结果如图 6-11 所示，可以看到，弹出栈的次序刚好和入栈的次序相反。

图 6-11　程序的运行结果，注意出栈的顺序

6.3.2 Stack 类

正如所期望的那样，Visual Basic 也提供了一个类 Stack 用来完成栈的运算。和前面的 List 类相类似，Stack 类是面向对象的泛型类。可以在声明时指定栈中的数据类型。Stack 类主要提供了以下方法。

- Clear。从 Stack 中移除所有对象。
- Contains。确定某元素是否在 Stack 中。
- Peek。返回位于 Stack 顶部的对象但不将其移除。
- Pop。移除并返回位于 Stack 顶部的对象。
- Push。将对象插入 Stack 的顶部。

还有一个重要的属性 Count 指出栈中元素的个数。注意并没有一个单独的判断栈是否为空的方法，可以通过 Count 属性是否大于 0 来判断。

例 6-6 使用 Stack 类实现例 6-5。

分析：要仔细注意使用现有的类来编程的格式和方法。

程序：

```
Module Module1

    Sub Main()
        '声明一个栈
        Dim stackExamp As New Stack(Of String)

        '初始栈，置为空栈
        stackExamp.Clear()

        '将一些字符串压入栈
        stackExamp.Push("西安")
        stackExamp.Push("交通")
        stackExamp.Push("大学")

        '只要栈不空，将栈里的字符串全部弹出栈并显示
        While stackExamp.Count > 0
            Console.Write(stackExamp.Pop() + " ")
        End While

    End Sub

End Module
```

程序的运行结果和例 6-5 是一样的，只是使用了 Stack 类只需要一个 Main 就可以了。下面作为栈的一个应用，再举一个例子。

例 6-7 检查表达式的括号是否匹配。

分析：从键盘输入一个表达式如(a+b)×(5+c)×((22–c)/23+56)，现在要检查输入的表达式中括号是否是匹配的。这可以用栈来实现。从左至右逐一读取表达式中的每一个字符，如果是左括号（则将其压入栈中，如果遇到一个右括号）则从栈中弹出一个左括号。

当处理完表达式的字符串时如果栈恰好也是空的，则表达式是匹配的。否则如果处理完表达式栈不空，或者表达式未处理完，需要出栈时栈是空的，此时可以断定，括号是不匹配的。

程序：

```
Module Module1

    Sub Main()
        Dim expression As String
        Console.WriteLine("请输入表达式")
        expression = Console.ReadLine()

        '声明一个栈
        Dim s As New Stack(Of Char)

        '初始栈，置为空栈
        s.Clear()

        Dim ch As Char
        '循环处理表达式中的每一个字符
        For i As Integer = 0 To expression.Length - 1
            ch = expression(i)
            If ch = "(" Then
                s.Push(ch)      '是左括号则压栈
            End If
            If ch = ")" Then
                If s.Count > 0 Then
                    s.Pop()     '是右括号则弹栈
                Else
                    '是右括号，但栈是空的，说明没有与之匹配的左括号
                    Console.WriteLine("括号不匹配")
                    End  '程序立即终止退出
                End If
            End If
        Next

        If s.Count = 0 Then
            Console.WriteLine("括号匹配")
        Else
            '栈非空，说明有左括号没有与之匹配的右括号
            Console.WriteLine("括号不匹配")
        End If

    End Sub

End Module
```

程序的运行结果如图 6-12 所示。

图 6-12 括号匹配检查的程序，给出了两种运行情形：匹配和不匹配

6.3.3 队列

和栈相类似，队列也是一种特殊的线性表，它只允许在表的前端（front）进行删除操作，而在表的后端（rear）进行插入操作。进行插入操作的端称为队尾，进行删除操作的端称为队头。队列中没有元素时，称为空队列。在队列这种数据结构中，最先插入的元素将是最先被删除的元素；反之最后插入的元素将是最后被删除的元素，因此，队列又称为“先进先出”（FIFO——First in First Out）的线性表。

队列可以用数组来存储，数组的上界即是队列所容许的最大容量。在队列的运算中需设两个索引下标：front，队头存放实际队头元素的前一个位置；rear，队尾存放实际队尾元素所在的位置。一般情况下，两个索引的初值设为 0 ，这时队列为空，没有元素。图 6-13 是一个队列的示意图。元素只能从队尾进入队列，只能从队头出队。也就是说要得到第 3 个元素 C，必须 A 和 B 先出队才可以。

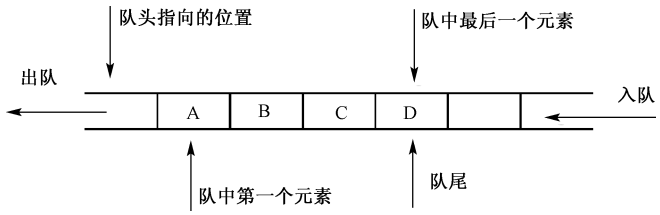


图 6-13 队列的示意图

当队头和队尾相等时，表示队列是空的，队尾到达了数组的上界，则队列是满的。图 6-14 是一个队列变化的示意图。图 6-14（a）中是这个队列中有两个元素的情形；图 6-14（b）是当 A 和 B 出队后，队列为空，此时队头等于队尾；图 6-14（c）队中依次进入了 3 个元素 C、D 和 E；图 6-14（d）C 是出队后队列的情形。

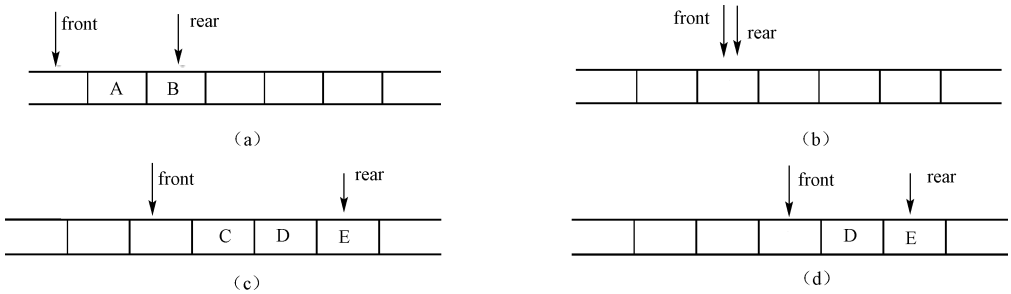


图 6-14 队列变化的示意图

一个队列，常用的操作如下：

- 求队列的长度 GetLength，得到队列中数据元素的个数。

- 判断队列是否为空 IsEmpty，如果队列为空返回 True，否则返回 False。
- 清空队列 Clear，使队列为空。
- 入队 EnQueue，将新数据元素添加到队尾，队列发生变化。
- 出队 DlQueue，将队头元素从队列中取出，队列发生变化。
- 取队头元素 GetFront，返回队头元素的值，队列不发生变化。

仔细观察图 6-14 的过程，会发现随着元素的出队和入队，队头和队尾均会不断地向后移动。当队尾移动到这个队列存储空间的最后一个位置时，如果还有元素要入队，则会发生溢出。因为队尾已经移到最后，没法再向后移动了。但实际上，队列中还是有空间的，因为有元素出队，也就是说，队头之前的空间是可以再用来存储数据的。如何利用空间呢？最直观的方法是将队列整个向前移动，但这样做效率并不高。一个较好的办法是将队列的头尾相连形成一个圆圈，这就是循环队列。循环队列的示意图如图 6-15 所示。当队尾和队头重叠时，队列为空还是满呢？我们约定，当队头和队尾相等时，队空。当队尾加 1 后等于队头时，队满。这样虽然浪费了一个存储空间（为什么？）但可以较为容易的区别队空和队满的情形。

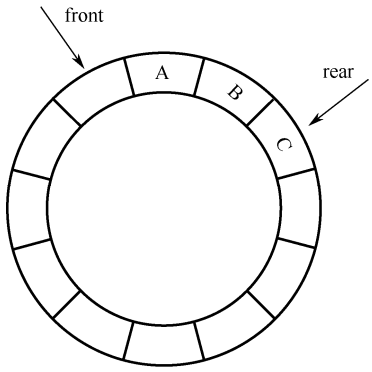


图 6-15　一个循环队列的示意图，空间可以反复利用

例 6-8　一个队列的实现。

分析：使用固定大小的数组，实现队列的简单操作。

程序：

```
Module Module1

    '定义队列结构
    Structure Queue
        Dim q() As Char
        Dim front As Integer
        Dim rear As Integer
    End Structure

    Sub Main()
        Const QueueMax As Integer = 30     '队列最大容量为 30
        Dim que As Queue
        ReDim que.q(QueueMax)     '重新指定队列数组的大小

        '清空队列
        Clear(que)

        '入队操作，3 个元素进入队列
        EnQueue(que, "A")
        EnQueue(que, "B")
        EnQueue(que, "C")
```

```
        '打印队列
        PrintQueue(que)

        '一个元素出队
        Dim ch As Char
        ch = DlQueue(que)

        '打印队列
        PrintQueue(que)

        '出队 1 个入队 3 个后打印队列
        ch = DlQueue(que)
        EnQueue(que, "D")
        EnQueue(que, "E")
        EnQueue(que, "F")
        PrintQueue(que)

    End Sub

    '清空队列，也就是头尾均为 0
    Sub Clear(ByRef q As Queue)
        q.front = 0
        q.rear = 0
    End Sub

    '入队操作，注意简单起见，并没有判断队满的情形
    Sub EnQueue(ByRef q As Queue, ByVal ch As Char)
        q.rear = q.rear + 1
        q.q(q.rear) = ch
    End Sub

    '出队操作，判断了队列是否为空
    Function DlQueue(ByRef q As Queue) As Char
        If Not IsEmpty(q) Then
            q.front = q.front + 1
            Return q.q(q.front)
        Else
            Return vbNullChar   '若队空，返回空字符（ASCII 码等于 0 的那个字符）
        End If
    End Function

    '从队头开始打印全部队列元素及队列长度
    Sub PrintQueue(ByVal q As Queue)
        For i As Integer = q.front + 1 To q.rear
            Console.Write(q.q(i) + " ")
        Next
        Console.Write("队列的长度是 {0} ", GetLength(q))
```

```
        Console.WriteLine()
    End Sub

    '判断队列是否为空
    Function IsEmpty(ByVal q As Queue) As Boolean
        If q.rear = q.front Then
            Return True
        Else
            Return False
        End If
    End Function

    '得到队列的长度
    Function GetLength(ByVal q As Queue) As Integer
        Return q.rear - q.front
    End Function
End Module
```

程序的运行结果如图 6-16 所示。

图 6-16　倒 6-8 队列的运行结果

例 6-9　循环队列。

分析：使用固定大小的数组，实现一个循环队列。这里的代码和例 6-8 基本是相同的，只是在入队、出队和打印等操作上注意对对头和队尾就队列的总长度取余。

程序：

```
Module Module1

    '定义队列结构
    Structure Queue
        Dim q() As Char
        Dim front As Integer
        Dim rear As Integer
    End Structure

    Const QueueMax As Integer = 30      '队列最大容量为 30
    Sub Main()
        Dim que As Queue
        ReDim que.q(QueueMax)           '重新指定队列数组的大小

        '清空队列
        Clear(que)
```

```
        '入队操作，3 个元素进入队列
        EnQueue(que, "A")
        EnQueue(que, "B")
        EnQueue(que, "C")

        '打印队列
        PrintQueue(que)

        '1 个元素出队
        Dim ch As Char
        ch = DeQueue(que)

        '打印队列
        PrintQueue(que)

        '出队 1 个入队 3 个后打印队列
        ch = DeQueue(que)
        EnQueue(que, "D")
        EnQueue(que, "E")
        EnQueue(que, "F")
        PrintQueue(que)

    End Sub

    '清空队列，也就是头尾均为 0
    Sub Clear(ByRef q As Queue)
        q.front = 0
        q.rear = 0
    End Sub

    '入队操作
    Sub EnQueue(ByRef q As Queue, ByVal ch As Char)
        If (q.rear + 1) Mod (QueueMax + 1) = q.front Then
            Console.WriteLine("队列已满，无法完成入队操作")
        Else
            q.rear = (q.rear + 1) Mod (QueueMax + 1)
            q.q(q.rear) = ch
        End If
    End Sub

    '出队操作，判断了队列是否为空
    Function DeQueue(ByRef q As Queue) As Char
        If Not IsEmpty(q) Then
            q.front = (q.front + 1) Mod (QueueMax + 1)
            Return q.q(q.front)
        Else
            Return vbNullChar   '若队空，返回空字符（ASCII 码等于 0 的那个字符）
```

```
        End If
    End Function

    '从队头开始打印全部队列元素及队列长度
    Sub PrintQueue(ByVal q As Queue)
        Dim i As Integer = q.front
        While i <> q.rear
            i = (i + 1) Mod (QueueMax + 1)
            Console.Write(q.q(i) + " ")
        End While

        Console.Write("队列的长度是 {0} ", GetLength(q))
        Console.WriteLine()
    End Sub

    '判断队列是否为空
    Function IsEmpty(ByVal q As Queue) As Boolean
        If q.rear = q.front Then
            Return True
        Else
            Return False
        End If
    End Function

    '得到队列的长度
    Function GetLength(ByVal q As Queue) As Integer
        If q.rear > q.front Then
            Return q.rear - q.front
        Else
            Return QueueMax + q.rear - q.front + 1
        End If
    End Function
End Module
```

6.3.4 Queue 类

和 Stack 类相似，Visual Basic 也提供了 Queue 类。该类实现了队列的常用算法，并且当队列容量不足时会自动增加队列的大小。下面是一些主要的方法：

- Clear。从 Queue 中移除所有对象。
- Contains。确定某元素是否在 Queue 中。
- CopyTo。从指定数组索引开始将 Queue 元素复制到现有一维 Array 中。
- Dequeue。移除并返回位于 Queue 开始处的对象。
- ToArray。将 Queue 元素复制到新数组。
- Enqueue。将对象添加到 Queue 的结尾处。
- Peek。返回位于 Queue 开始处的对象但不将其移除。

和 Stack 类一样，Queue 类的唯一一个重要的属性是 Count，表示队列中含有元素的个数。

例 6-10 使用 Queue 类实现例 6-9。

程序：

```
Module Module1

    Sub Main()
        Dim que As New Queue(Of Char)

        que.Clear()   '清空队列

        '入队操作，3 个元素进入队列
        que.Enqueue("A")
        que.Enqueue("B")
        que.Enqueue("C")

        '打印队列
        PrintQueue(que)

        '1 个元素出队
        Dim ch As Char
        ch = que.Dequeue()

        '打印队列
        PrintQueue(que)

        '出队 1 个入队 3 个后打印队列
        ch = que.Dequeue()
        que.Enqueue("D")
        que.Enqueue("E")
        que.Enqueue("F")
        PrintQueue(que)

    End Sub


    '从队头开始打印全部队列元素及队列长度
    Sub PrintQueue(ByVal q As Queue(Of Char))
        '先将队列中的所有元素复制到一个数组中
        Dim queArray(q.Count - 1) As Char
        q.CopyTo(queArray, 0)
        For Each member As Char In queArray
            Console.Write(member + " ")     '输出数组的每一个元素
        Next
        Console.Write("队列的长度是 {0} ",q.Count)
        Console.WriteLine()
    End Sub

End Module
```

6.4 图和树

前几节讲述了一些线性结构的数据，而图和数则是非线性的数据结构。同时，现实中的很多问题也是用线性数据结构而无法描述的，需要借助非线性的数据结构来描述。

6.4.1 图的基本概念

1736 年，著名数学家欧拉（Euler）发表了著名论文《柯尼斯堡七座桥》的论文中，首先使用图的方法解决了柯尼斯堡七桥问题，因而欧拉也被誉为图论之父。这个问题是基于一个现实生活中的事例：当时东普鲁士柯尼斯堡（Königsberg，今日俄罗斯加里宁格勒）市区跨普列戈利亚河（Pregel）两岸，河中心有两个小岛。小岛与河的两岸有 7 座桥连接。于是，7 座桥将 4 块陆地连接了起来，如图 6-17 所示。而城里的居民想在散步的时候从任何一块陆地出发，经过每座桥 1 次且仅经过 1 次最后返回原来的出发点。当地的居民和游客做了不少尝试，却都没有成功，而欧拉最终解决了这个问题并断言这样的回路是不存在的。

欧拉在解决问题时，用 4 个结点来表示陆地 A、B、C 和 D，凡是陆地间有桥连接的，便在 2 点间连一条线，于是图 6-17 转换为图 6-18。

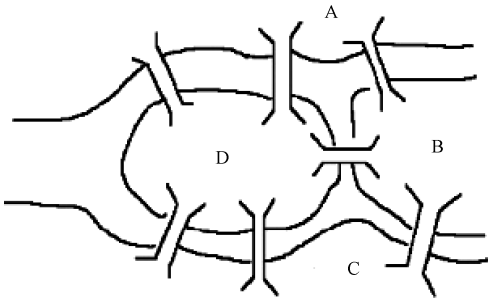


图 6-17　柯尼斯堡七桥问题示意图

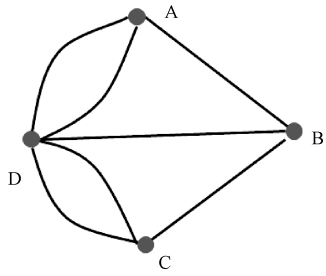


图 6-18　柯尼斯堡七桥问题抽象为图后的表示

而此时，问题则转化为从图 6-18 中的 A、B、C、D 任一点出发，通过每条边一次且仅一次后回到原出发点的回路是否存在？欧拉断言了这个回路是不存在的，理由是从图 6-18 中的任一点出发，为了能够回到原出发点，则要求与每个点关联的边数均为偶数。这样才能保证从一条边进入某点后可以从另外一条边出来。而图 6-18 中的 A、B、C、D 全部都与奇数边关联，因而回路是不存在的。

而由上面的例子我们也看到，图（Graph）是由结点或称顶点（Vertex）和连接结点的边（Edge）所构成的图形。使用 V(G)表示图 G 终所有结点的集合，E(G)表示图 G 中所有边的集合。则图 G 可记为<V(G),E(G)>或<V,E>。有 n 个顶点和 m 条边的图记为（n,m）图或称为 n 阶图。

例 6-11　4 个城市 v_1、v_2、v_3 和 v_4。v_1 和其他 3 个城市都有道路连接，v_2 和 v_3 之间有道路连接，画出图并用集合表示该图。

显然结点集合 V={v_1，v_2，v_3，v_4}，边集合 E={v_1 和 v_2 之间的边，v_1 和 v_3 之间的边，v_1 和 v_4 之间的边，v_2 和 v_3 之间的边}。画出的图如图 6-19 所示。

更一般的，边可以用结点对来表示，或者说用结点 V 的矢量积来表示，即

$$V=\{v_1,v_2,v_3,v_4\}$$

$$E=\{(v_1,v_2),(v_1,v_3),(v_1,v_4),(v_2,v_3)\}$$

如果边不区分起点和终点，这样的边称为无向边。所有边都是无向边的图称为无向图，如图 6-19 就是一个无向图。反之，若边区分起点和终点，则为有向边，所有边都是有向边的图称为有向图。在图中，有向边使用带有箭头的线段表示，从起点指向终点。在集合中则用有序对<v_1,v_2>来表示，图 6-20 是一个示例。

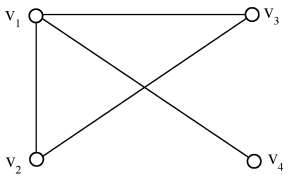


图 6-19　例 6-11 中的图

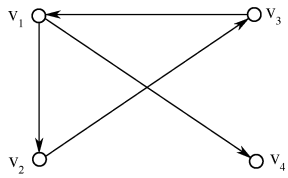


图 6-20　一个有向图的示例

在图 6-20 中，即

$$V=\{v_1，v_2，v_3，v_4\}$$

$$E=\{<v_1,v_2>,<v_1,v_4>,<v_3,v_1>,<v_2,v_3>\}$$

结点的度则是指和结点关联的边的个数。如在图 6-19 中，v_1 的度是 3，v_2 和 v_3 的度是 2，v_4 的度是 1。对于有向图，则区分为出度和入度，由结点指向外的边的个数为出度，反之为入度。如图 6-20 中，v_1 的出度为 2，入度为 1，v_4 的出度为 0，入度为 1。

图在计算机中如何存储，则是人们普遍关心的一个问题。简单的方法是将图用一个二维矩阵来表示，这样的矩阵通常称为邻接矩阵。在此我们不系统讨论，仅以图 6-20 的存储为例来说明。

例 6-12　将简单有向图（图 6-20）以邻接矩阵的方式存储到计算机中。

要以邻接矩阵的方式存储，首先需要对结点指定一个次序。在此，我们就以结点的下标从小到大为序，排列为 v_1，v_2，v_3，v_4。然后使用一个 4×4 的矩阵来存储该图，矩阵中的元素只有 2 个取值：0 或者 1。对于两个结点 v_i 和 v_j，若 v_i 和 v_j 之间存在一条边，则对应的矩阵元素 $a_{ij}=1$，反之则为 0。图 6-20 示例的有向图存储矩阵如下图 6-21 所示。

容易看出，矩阵中 1 的个数对应图中边的个数，而对角线的元素则全为 0。

	v_1	v_2	v_3	v_4
v_1	0	1	0	1
v_2	0	0	1	0
v_3	1	0	0	0
v_4	0	0	0	0

图 6-21　存储的邻接矩阵

6.4.2　带权图和最短路径

图的问题异常的复杂，甚至是一门完整的学科——图论。在此无法对图有一个完整系统的讨论。而为了使读者对图有进一步的认识，作为一个例子，简单介绍带权图及最短路径的算法，并以此结束对图的讨论。

在处理有关图的实际问题时，往往有值的存在，如公里数、运费、城市、人口数及电话部数等。一般这个值称为权值，在图中，将每条边都有一个非负实数对应的图称为带权图或赋权图。这个实数称为这条边的权。根据不同的实际情况，权数的含义可以各不相同。

例如，可用权数代表两地之间的实际距离或行车时间，也可用权数代表某工序所需的加工时间等。如图 6-22 所示便是一个带权图。

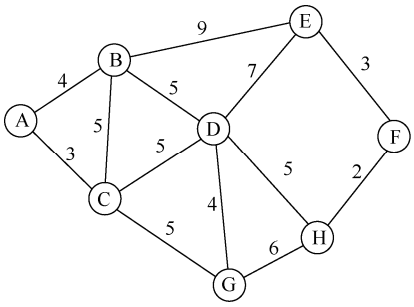


图 6-22　带权图

在图 6-22 所示的无向带权图求最短路径是一个经常遇到的很实际的问题。假设在图 6-22 中的 A 到 G 点表示 8 个村庄，边表示村庄之间的道路。边上的权值表示距离。现在的问题是从 A 到 F 最短的距离是多少？

求最短路的算法是 E • W • Dijkstra 于 1959 年提出来的，这是至今公认的求最短路径的最好方法，我们称它为 Dijkstra 算法。假设给定带权图 G，要求 G 中从 v0 到 v 的最短路径，Dijkstra 算法的基本思想如下：

将图 G 中结点集合 v 分成两部分：一部分称为具有 P 标号的集合，另一部分称为具有 T 标号的集合。结点 a 的 P 标号是指从 v0 到 a 的最短路的路长；而结点 b 的 T 标号是指从 v0 到 b 的某条路径的长度。Dijkstra 算法中首先将 v0 取为 P 标号结点，其余的结点均为 T 标号结点，然后逐步地将具有 T 标号的结点改为 P 标号结点，当目的结点也被改为 P 标号时，则找到了从 v0 到 v 的一条最短路径。下面通过一个例子给出实际的算法步骤。

例 6-13　计算图 6-22 所示的带权图中，从 A 点到 F 点的最短路径。

（1）首先，将起点 A 划归为 P 标号集合，其余的结点均为 T 结点。A 到 A 的距离为 0，所以 A 的 P 标号为 0。

（2）更新 T 中结点到 A 的距离，如和 A 相邻（有边连接）则就是边的权值。如和 A 没有直接的边连接则距离是无穷大。

（3）在 T 中找到一个值最小的结点，并将其划归到 P 集合。此时，计算的结果如图 6-23 所示（C 结点进入 P 集合）。

（4）根据新进入的 C 结点，更新与 C 相连的结点的值。新值等于 C 的 P 结点值加上到与其相连的结点的距离（边的权值）。更新的算法是如果新值小于原有的值，则用新的值取代，否则保持原有值不变。

（5）重复步骤（3）和（4）直到目标点进入 P 集合。

图 6-24～图 6-29 演示了这一过程。

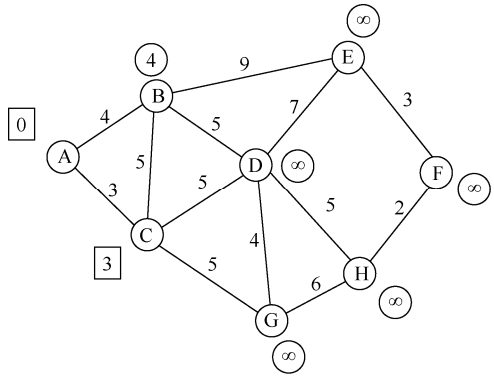


图 6-23　A 到 B 的距离为 4，到 C 的距离为 3，到其余结点的距离为无穷大

由于 C 结点的值最小，因而 C 进入 P 集合（P 集以方框表示，T 集用圆圈表示）

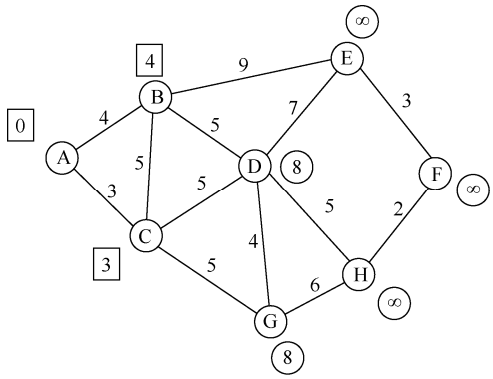


图 6-24　结点 C 进入 P 集合后，到 B 的距离为 3+5=8 大于 B 原来的 4，因而 B 的值不变

而到 D 和 G 的值均为 8，均小于原来的无穷大，因而用 8 取代原来的值。之后，在 T 中，B 的值为 4 最小，B 进入 P 集合。

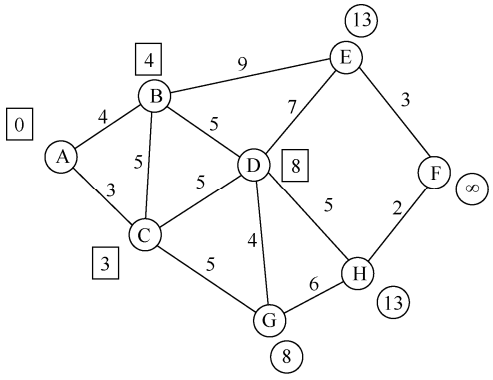


图 6-25　结点 B 进入后更新与 B 连接的 D 和 E 值

其中，D 的值不变，E 为 13。此时 D 和 G 均有最小值 8，任取一进入 P，在此是取的是 D。然后又更新了 H 的值。G 的原值小于 8+4，因而保持不变。

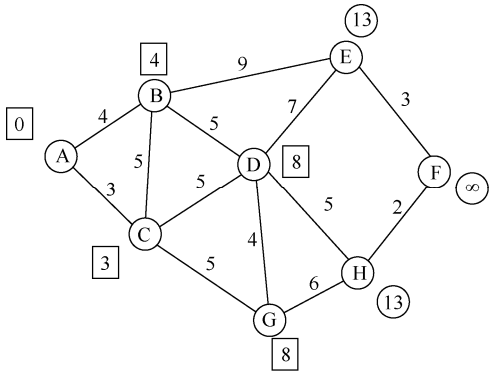


图 6-26　结点 G 的值最小进入 P，H 的值未变

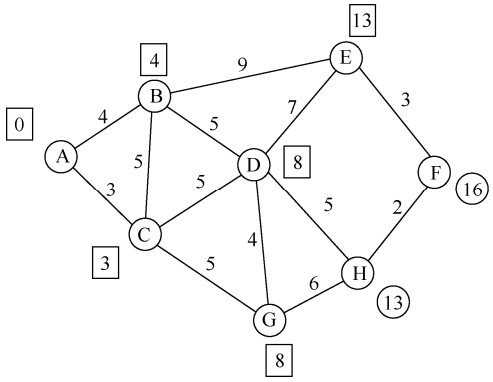


图 6-27　任选 E 进入 P，F 值变为 16

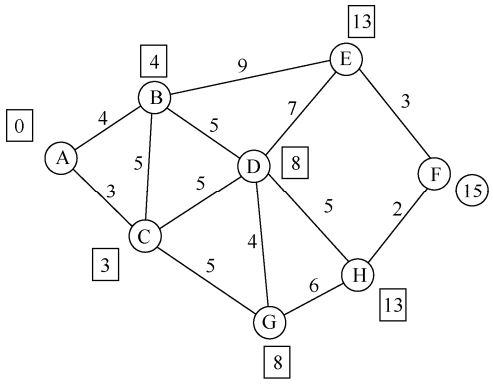


图 6-28　结点 H 进入 P，F 的值变为 15

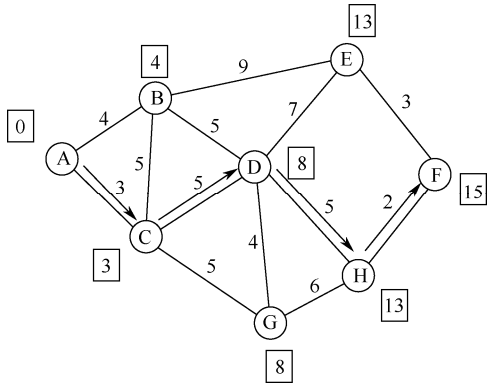


图 6-29　终点 F 进入 P 集合，运算结束

从 A 到 F 的最短距离为 15。而事实上，对于每一个 P 中的结点值，计算出了从 A 到该结点的最短距离，如到 E 的最短距离为 13。而找到最短路径的方法是用 F 点的 P 值减去边的权值，倒推回 A 点。如 F 的值 15–2=13 和 H 吻合，而不是 E（因为 15–3=12 不等于 E 的 13）。

6.4.3　树的基本概念

树可以看作是一个特殊的有向图。对于一个有向图，如果

（1）存在一个特殊的结点 r，其入度等于 0。

（2）除了 r 外的其他结点的入度均为 1。

（3）R 到图中其他结点均有路可达。

满足这样的图称为树。其中入度为 0 的结点称为根，出度为 0 的结点称为叶子。出度不为 0 的结点称为分枝点，如图 6-30 所示。

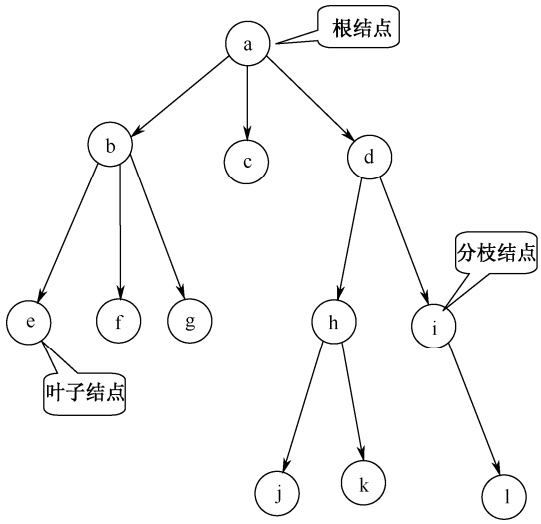


图 6-30　树

在画树的图的时候，由于所有的箭头方向都是一致的，所以箭头常常省略，如图 6-31 所示。树是有层次的，指的是从根到该结点的距离。称距根最远的叶子的层数为树的高度。图 6-31 的树的高度为 3。同一层次之间的结点称为兄弟，上一层次的为父亲，下一层次是儿子，如图 6-31 中对 h 结点的描述。

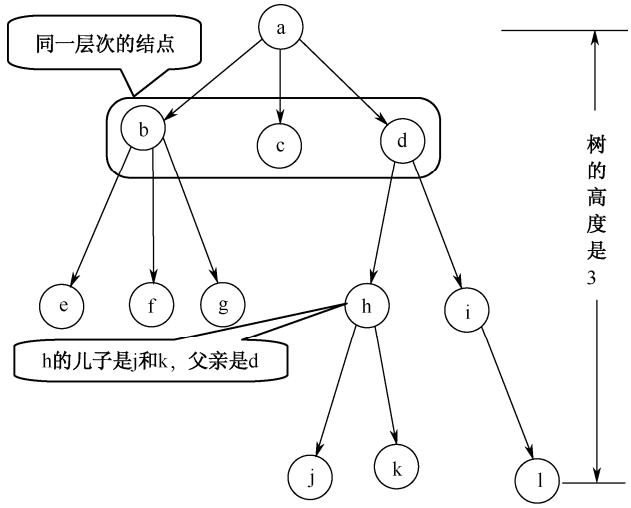


图 6-31　树的高度及层次关系

对于一棵树而言，若所有结点的入度均不大于 m，则称此树为 m 叉树。如果每个结点的入度都相等且都等于 m，则称此树为完全 m 叉树。在计算机学科经常应用的是二叉树。

6.4.4 二叉树

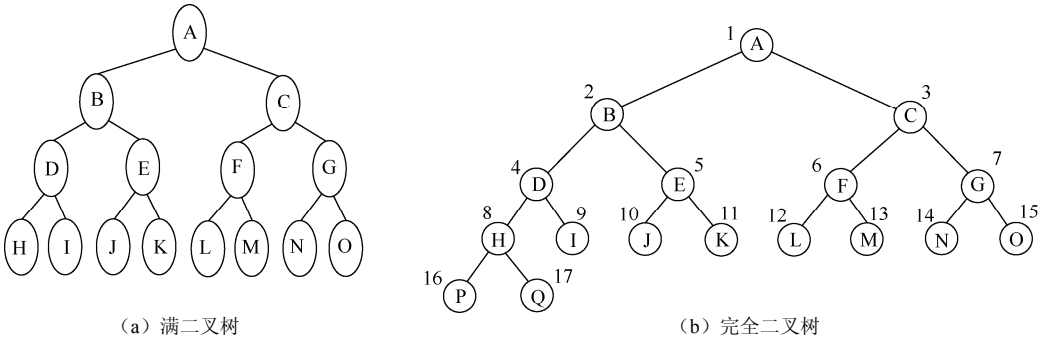
二叉树是每个结点最多有两个子树的树结构。通常子树称为“左子树”（Left Subtree）和“右子树”（Right Subtree）。二叉树的每个结点至多只有两棵子树（不存在出度大于 2 的结点），二叉树的子树有左右之分，次序不能颠倒。

二叉树具有如下性质。

- 二叉树的第 i 层至多有 2^{i-1} 个结点。
- 深度为 k 的二叉树至多有 2^{k-1} 个结点。
- 二叉树的结点个数可以为 0。
- 二叉树的结点有左、右之分。

一棵深度为 k，且有 2^{k-1} 个结点的二叉树，称为满二叉树（Full Binary Tree）。这种树的特点是每一层上的结点数都是最大结点数。

而对于深度为 K 的，有 N 个结点的二叉树，当且仅当其每一个结点都与深度为 K 的满二叉树中编号从 1 至 n 的结点一一对应时称为完全二叉树（Complete Binary Tree）。也就是说，若一棵二叉树至多只有最下面的两层上的结点的度数可以小于 2，并且最下层上的结点都集中在该层最左边的若干位置上，则此二叉树成为完全二叉树。具有 n 个结点的完全二叉树的深度为 $\log_2 n+1$。深度为 k 的完全二叉树，至少有 2^{k-1} 个结点，至多有 2^k-1 个结点。图 6-32（a）和图 6-32（b）分别是满二叉树和完全二叉树的示例。



（a）满二叉树　　（b）完全二叉树

图 6-32　满二叉树和完全二叉树

*6.4.5 树的遍历

树的遍历是树的一种重要的运算。遍历是指对树中所有结点的信息的访问，即依次对树中每个结点访问一次且仅访问一次。树的遍历有两种方式：先根的遍历方式，即先访问树的根结点，然后依次先根的访问根的每一颗子树；后根的遍历方式，即依次后根的访问根的每一颗子树，最后访问根结点。

对于图 6-31 的树，采用先根的方式，结点访问的次序依次为

```
a b e f g c d h j k i l
```

采用后根的方式，结点访问的次序依次为

```
e f g b c j k h l i d a
```

习题

1．什么是数据的线性存储结构？什么是数据的非线性存储结构？

2．简述线性表的操作。

3．假设电话号码本由人名和一个电话号码组成，设计一个线性表，存储 7 个人的电话号码簿。

4．设栈 S 中存储的是字符数据，自栈底到栈顶依次为 A、C、D。经过两次出栈操作并将 E 压入栈，此时栈中的数据是什么？

5．使用栈，检查表达式（2+3）×a×（3+b）/（2×（12+8）的括号是否匹配。

6．编写程序，输入一行文本，然后使用栈逆序显示该行文本。

7．编写程序，用栈来判断一个字符串是否为回文（即顺读和倒读都相同的字符串）。程序忽略字符串中的大小写，空格和标点符号。

8．使用 LinkedList 类实现第 3 题的电话簿，打印该号码簿。然后删去第 2 个和最后一个结点的数据再次打印号码簿。

9．设计一个队列，将整数 3，4，5 进入队列，打印该队列，将队列的前两个元素出队，随后将 11 和 12 入队，再次打印队列。

10．对于图 6-15 的循环队列，在该图的基础上，将 1、2、3、4、5 入队，并将两个元素出队后，画出队列目前的状态。

11．将书中图 6-22 的带权图使用邻接矩阵的方式存储到计算机中，试写出该矩阵（提示：是一个对角线元素为 0 的对称矩阵）。

12．*试描述使用邻接矩阵 7（在 11 题的基础上）计算最短路径时的算法。

13．*试编写程序（在第 11 题和第 12 题的基础上）实现图 6-22 最短路径的计算。

第 7 章　算法分析与设计

引言

算法可以理解为有基本运算及规定的运算顺序所构成的完整的解题步骤。或者看成按照要求设计好的有限的确切的计算序列，并且这样的步骤和序列可以解决一类问题。算法是解题方案的准确而完整的描述，是一系列解决问题的清晰指令，算法代表着用系统的方法描述解决问题的策略机制。也就是说，能够对一定规范的输入，在有限时间内获得所要求的输出。如果一个算法有缺陷，或不适合某个问题，执行这个算法将不会解决这个问题。不同的算法可能用不同的时间、空间或效率来完成同样的任务。一个算法的优劣可以用空间复杂度与时间复杂度来衡量。

教学目的

- 掌握常用的查找算法。
- 掌握常用的排序算法。
- 掌握常用查找和排序算法的时间复杂度分析。
- 对常用算法有一定的了解。

7.1　算法的基本概念

算法（Algorithm）一词来源于阿拉伯数学家 AIKhowarizmi 编写出版的《波斯教科书》（Persian Textbook），书中概括了进行算术四则运算的法则。后来的《韦氏新世界词典》将其定义为“求解某种问题的任何专门的方法”。

对复杂的系统性问题，算法设计是需求分析建立起相应的业务模型和数学模型，以及模块设计确定了相应的系统结构和数据结构的基础上，对模块功能的进一步细化，是求解问题的方法的描述。算法设计包括确定算法的控制结构（顺序、循环或选择），以及实现的具体步骤和操作。算法设计的正确与否、精练与否，决定了程序编码的正确性和有效性。

利用算法求解问题的一般思路如图 7-1 所示。

图 7-1　利用算法求解问题的一般思路

算法是一个有穷规则的集合。它表现为一个解决某个特定类型问题的运算序列。通俗

地讲，算法指解决某个问题的一系列步骤或方法，如果遵循它就可以完成一项特定的任务。算法是定义在逻辑结构上的操作，是独立于计算机的，而它的具体实现则是在计算机上进行的。

前面曾给出了“统计某个班学生的英语和数学这两门课的成绩，找出合计最大并且没有不及格的学生”的算法描述。回顾该算法，我们可以看出算法应具有如下特性：

- **有穷性**。一个算法必须是在执行有穷步后结束，且每一步都能在有限的时间内完成。即一个算法所包含的计算步骤和时间都是有限的。
- **确定性**。算法的每一个步骤都必须具有确切的定义，即算法中所有有待执行的动作，都必须有严格的、毫不含混的规定，不能有歧义。
- **能行性（或称可行性）**。算法中所有有待实现的运算都必须是能够精确执行的，且用纸和笔做有穷次即可完成。算法的执行者甚至不需要掌握算法的含义即可根据算法的每个步骤要求进行操作，并最终得出正确的结果。
- **输入**。一个算法应该有 0 个或多个输入。
- **输出**。一个算法应该有 1 个或多个输出。

例 7-1　设计求 2+4+6+8+…+10 000 的算法。

算法描述如下：

步骤 1：让变量 SUM=0。

步骤 2：让变量 J=2。

步骤 3：计算 SUM+J，结果仍放在 SUM 中，即让 SUM=SUM+J。

步骤 4：让 J=J+2。

步骤 5：如果 J 不大于 10 000，返回执行步骤 3，否则执行下一步。

步骤 6：输出结果 SUM 的值。

在例 7-1 中，步骤 3 至步骤 5 重复执行了 4999 次，这就是循环结构。另外，步骤 5 是一个逻辑判断，判断的结果导致两种可能的执行流程，一种是向上循环执行，另一种是向下执行。这就是选择结构。

对于求 2+4+6+8+…+10 000 的算法，还可以有其他计算方法求解。例如，利用公式来计算，即只要计算(1+5000)×5000。这样一来，算法就只有三步：先计算加法，再计算乘法，最后输出结果。

可见，算法设计是非常灵活的，对同一个问题可以有不同的算法描述。但不同的算法可能有不同的效率。对于复杂问题，算法就更重要了。要在保证求解问题正确的前提下，尽可能地追求算法的效率，也就是要尽可能地设计出复杂度低的算法。

7.2　算法的描述方法

算法的表示方法有多种，最简单的就是自然语言表示法。除此之外，常用的描述方法还有伪代码、流程图等。

7.2.1　算法的自然语言描述

自然语言就是人们在日常生活中使用的语言，如汉语、英语、日语和俄语等。对初学

者来说，用自然语言描述算法最为直接，没有语法语义障碍，容易理解。但用自然语言描述算法文字冗长，不够简明，尤其会出现含义不太严格，要根据上下文才能判断出正确含义的情况。

例 7-2 描述求任意两个正整数的最大公因数的算法。

我们先来看一下著名的欧几里德算法。古希腊数学家欧几里德曾给出了求解两个数的最大公因子的算法，描述如下：

步骤 1：如果 p＜q，交换 p 和 q;

步骤 2：求出 p/q 的余数 r;

步骤 3：如果 r=0，则 q 就是所求的结果；否则反复做如下工作:

令 p=q，q=r，重新计算 p 和 q 的余数 r，直到 r=0 为止。q 就是原来的两正整数的最大公因数。

下面将欧几里德算法进一步细化：

步骤 1：输入两个正整数，分别放在变量 P 和 Q 中。

步骤 2：如果 P＜Q，则交换 P 和 Q 的值（即让 R=P，P=Q，Q=R）。

步骤 3：将 P/Q 的余数放在 R 中（即让 R=P/Q 的余数）。

步骤 4：如果 R 等于 0，则执行第 6 步，否则执行第 5 步。

步骤 5：让 P=Q，Q=R，执行第 3 步。

步骤 6：输出 Q 的值。

7.2.2 算法的伪代码描述

伪代码（Pseudo Code）介于自然语言和计算机语言之间，用你熟悉的计算机语言的语句加上自然语言构成（尽可能地融入编程语言的函数和语法），基本上可以随心所欲地写，例如，输入并比较两个学生成绩的过程可用类 C 语言描述如下(对一个班的成绩处理要更复杂一些)：

```
cout << 请输入学生姓名、学号、英语成绩、数学成绩
cin >>姓名 1 >>学号 1 >> 英语成绩 1 >>数学成绩 1
合计 1=英语成绩 1 +数学成绩 1
cout << 请输入学生姓名、学号、英语成绩、数学成绩
cin >>姓名 2 >>学号 2 >> 英语成绩 2 >>数学成绩 2
合计 2=英语成绩 2+数学成绩 2
if（合计 1> 合计 2）
if（英语成绩 1>0 并且 数学成绩 1>0）
     cout << 姓名 1 << 学号 1
else
   if（英语成绩 2>0 并且 数学成绩 2>0）
      cout << 姓名 2 << 学号 2
```

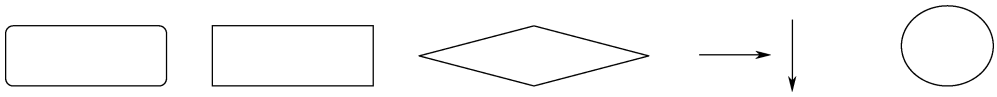
这个算法是有缺陷的，你看出来了吗?

由于此例子相当简单，接触过 C 语言的人可能看出，这基本上就接近程序本身了。

7.2.3 算法的流程图描述

流程图（Flow Chat），是用几种几何图形、线条和文字来表示不同的操作和处理步骤。

用流程图表示算法，形象直观，简洁清晰，易于理解。美国国家标准化协会（American National Standard Institute，ANSI）规定了常用流程图符号如图 7-2 所示。



（a）起始框　（b）处理框　（c）判断框　（d）流向线　（e）连接点

图 7-2　常用流程图符号

上述“输入并比较两个学生成绩”的算法可以用流程图方式描述如图 7-3 所示。由图中可以比较清楚地看出该算法描述存在的问题，即存在没有输出信息的可能。而算法的基本特性之一是要求至少有一个输出。

请考虑如何修改？

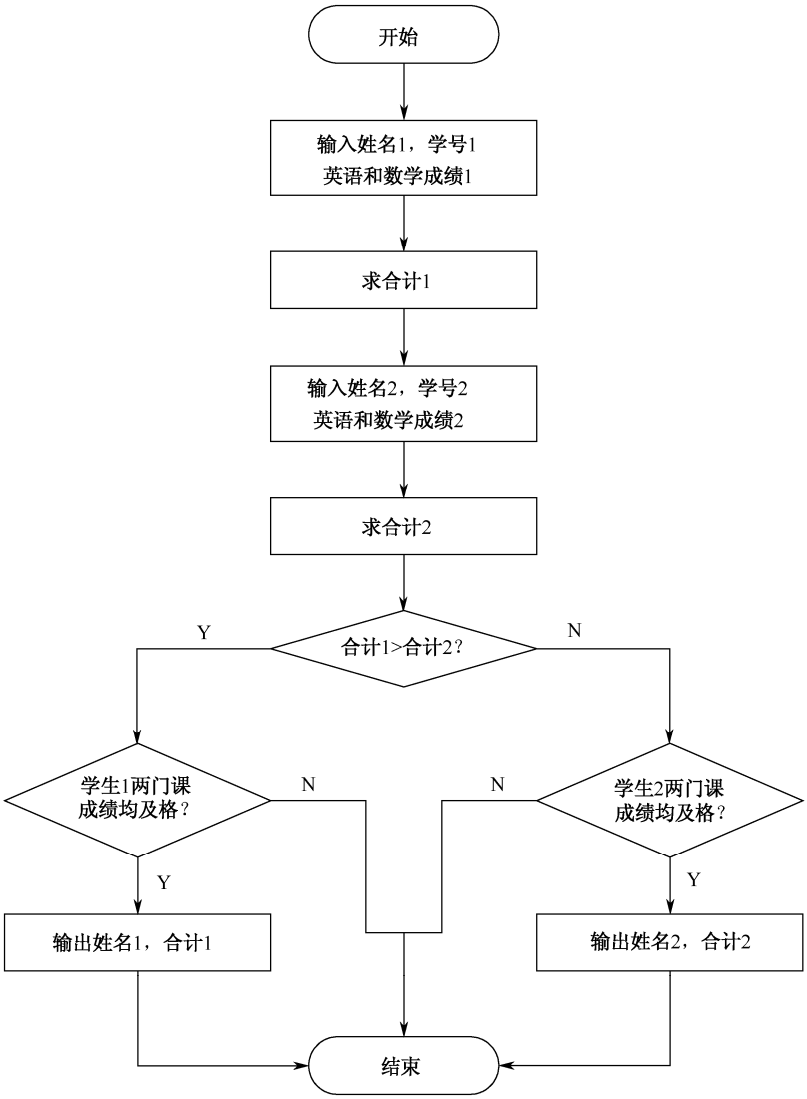


图 7-3　输入并比较两个学生成绩的算法流程

例 7-3 用流程图描述欧几里德算法。

解 欧几里德算法流程如图 7-4 所示。需要注意的是，图中的“R=P，P=Q，Q=R”处理框也可以分解成三个处理框。

可以比较一下用流程图表示的欧几里德算法与例 7-1 中用自然语言描述的算法，不难发现，流程图描述的算法逻辑清晰，直观形象，易于理解。

关于流程的详尽程度，并没有一个绝对统一的标准，因此，算法设计的结果并不唯一。对于初学者来说，只要能正确求解问题就可以。

在画流程图（即设计算法）时，往往会出现一张纸由上而下画满了，但算法描述还未结束，这时候就要将连接点符号画在纸张的底部，然后在另一张白纸的头部也画同样的连接点符号。这就意味着两张算法流程图被拼接起来，形成一幅完整的流程图。当然也会出现纸张左右画满的情况，这时候也需要用连接点符号。判断框有一个入口两个出口，两个出口的条件总是截然相反的，一个若代表条件成立，则另一个代表条件不成立。只要在两个出口流向线之一的旁边标注清楚即可。

下边再来看一个利用流程图描述算法的示例。

例 7-4 用流程图 7-5 描述求解“1–1/2+1/3–1/4+ 1/5–1/6+…+1/99–1/100”的算法。

解

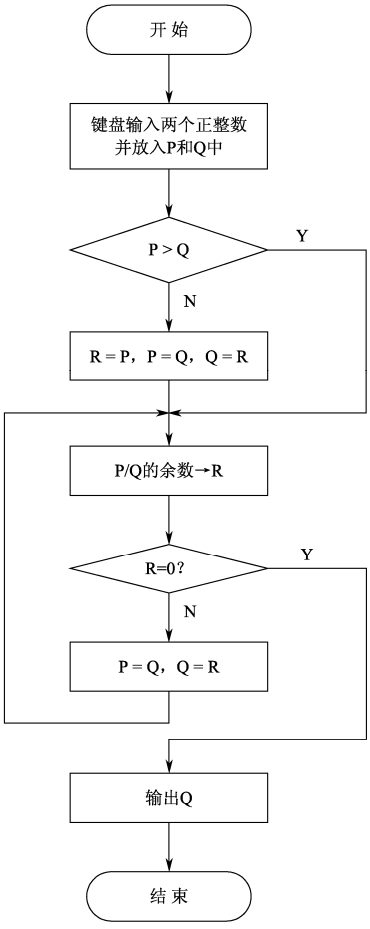


图 7-4 欧几里德算法流程图

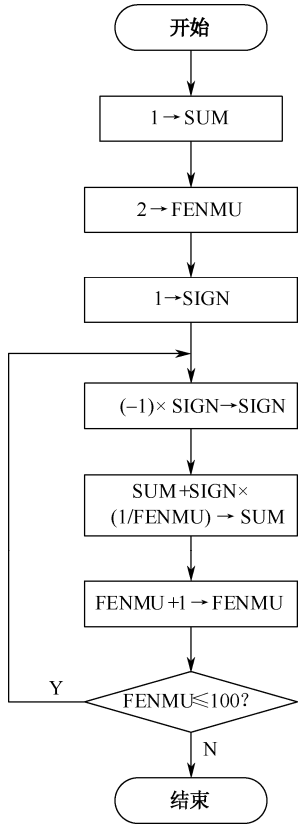


图 7-5 例 7-5 算法流程

7.3　算法的复杂性评价

解决同样一个问题可以用不同的算法。一个算法的质量优劣将影响到算法乃至程序的效率。

对算法的分析和评价一般应考虑正确性、可维护性、可读性、运算量及占用存储空间等诸多因素。通常，在算法“正确性”的前提下，评价一个算法的主要指标如下：

① 算法实现所消耗的时间，即时间复杂度。

② 算法实现所消耗的存储空间，即空间复杂度。

③ 算法应易于理解、易于编码、易于调试等。

7.3.1　算法的时间复杂度

1．时间频度

要确定一个算法执行所耗费的时间，最直接的方法就是测试。但我们不可能也没有必要对每个算法都上机测试，只需知道哪个算法花费的时间多，哪个算法花费的时间少就可以了。

假设可以知道算法中每一条语句执行一次所需的平均时间，则

算法运行所需的时间=语句执行一次所需的平均时间×语句执行次数　　（7-1）

但语句执行一次所需的平均时间取决于计算机 CPU 的主频、是否分时系统、编译系统的效率和优化程度、输入/输出速度等不确定因素。而一般来说，一个算法中语句的执行次数是确定的。由式（7-1）知，一个算法花费的时间与算法中语句的执行次数成正比，即算法中语句执行次数越多，它花费的时间就越多。一个算法中的语句执行次数称为语句频度或时间频度，记为 T(n)。

2．时间复杂度

时间频度中的 n 称为问题的规模（大小），如 n 条语句指令、n 个子程序、n 个功能模块等所需的执行时间。当 n 不断变化时，时间频度 T(n)也会不断变化。但有时我们想知道它变化时呈现什么规律。为此，我们引入时间复杂度的概念。

一般情况下，算法中基本操作重复执行的次数是问题规模 n 的某个函数，用 T(n)表示，若有某个辅助函数 F(n)，使得当 n 趋近于无穷大时，得

$$\lim_{n\to x}\frac{T(n)}{F(n)}=M\text{（正常数）}$$

则称 F(n)是 $T(n)$的同数量级函数。记作 T(n)=O(F(n)),称 O(F(n))为算法的渐进时间复杂度，简称时间复杂度。

按数量级增序排列，常见的几种时间复杂度有常数阶 O(1)、线性阶 O(n)、对数阶 O(log n)、线性对数阶 O(nlog n)、平方阶 $O(n^2)$、立方阶 $O(n^3)$、k 次方阶 $O(n^k)$、指数阶 $O(2^n)$等。随着问题规模 n 的不断增大，上述时间复杂度也就不断增大，算法的执行效率就越低。

在各种不同算法中，若算法中语句执行次数为一个常数，则时间复杂度为 O(1)。而下面的 Basic 语句可认为是 $O(n^2)$时间复杂度：

```
For i=0 To n-1
   For j=0 To n-1
        A(i,j)=0
   Next
Next
```

值得指出的是，在时间频度不相同时，时间复杂度有可能相同。例如，$T(n)=n^2+3n+4$与 $T(n)=4n^2+2n+1$。它们的频度不同，但时间复杂度相同，都为 $O(n^2)$。

不同的复杂度函数之间的对比如表 7-1 所示。

表 7-1　复杂度函数值对比

logn	n	nlogn	n^2	2^n
0	1	0	1	2
1	2	2	4	4
2	4	8	16	16
3	8	24	64	256
4	16	64	256	65 535
5	32	160	1 024	4 294 967 296

7.3.2　算法的空间复杂度

算法的空间复杂度是算法在计算机内执行时所需存储空间的度量。一个算法的实现所占用的存储空间大致有这样三个方面：一是指令、常数、变量所占用的存储空间；二是输入数据所占用的存储空间；三是算法执行时必需的辅助空间。前两个空间是计算机运行时所必须的。因此，我们把算法在执行时所需的辅助空间的大小作为分析算法空间复杂度的依据。

与算法时间复杂度的表示一致，我们也将算法中所用辅助空间大小的数量级来表示算法的空间复杂度，仍然记为 O(n)。常见的几种空间复杂度有 O(logn)，O(n)，$O(n^2)$，$O(2^n)$等。

事实上，对一个问题的算法实现，时间复杂度和空间复杂度往往是相互矛盾的，要降低算法的执行时间就要以使用更多的空间为代价，要节省空间就可能要以增加算法的执行时间作代价，两者很难兼顾。因此，只能根据具体情况有所侧重。

7.4　查找算法

“数据查找”是我们经常碰到的问题之一。查找，就是根据给定的关键字值（就是数据元素中可以唯一标识一个数据元素的数据项，如学生的学号、居民身份证号码等），在一组数据中确定一个其关键字值等于给定值的数据元素。若存在这样的数据元素，则称查找是成功的；否则称查找不成功。一组待查数据元素的集合又称为查找表。在查找过程中，查找表一旦建立不再改变的查找称为静态查找。反之则是动态查找。在此仅讨论静态查找。

7.4.1　顺序查找

顺序查找是最普通也是最简单的查找技术。其基本思想：从数组中的第一个元素开始，逐个把元素的关键字值和给定值比较，若某个元素的关键字值和给定值相等，则查找成功；

否则，若直至第 n 个记录都不相等，说明不存在满足条件的数据元素，查找失败。

顺序查找算法的伪代码描述如下：

```
从第一个元素开始
    while（）{
       比较当前元素的关键字值与所需关键字值
       if（找到了）
          返回当前元素
       if（所有元素找过了）
          返回查找失败
       下一个元素
       }
```

在该算法中，执行频率最高的是 while 语句。当查找表中元素个数 n 很大时，其平均查找长度 ASL=(n+1)/2。即每次查找平均要比较一半数据元素。

例 7-5　在整数数组内顺序查找。

分析：从头开始，逐一匹配，找到为止。

程序：

```
Module Module1

    Sub Main()
        Dim A() As Integer = {13, 67, 89, 2, 15, 99, 77, 56, 34}
        Dim k = SqSearch(A, 15)   '搜索 15
        If k >= 0 Then
            Console.WriteLine("查找的数据在数组中的位置为：" + k.ToString())
        Else
            Console.WriteLine("数据没有找到")
        End If
    End Sub

    '该函数在数组中搜索特定的数，如果找到，则返回数据在数组中的位置
    '否则返回-1
    Function SqSearch(ByVal A() As Integer, ByVal key As Integer) As Integer
        For i As Integer = 0 To A.GetLength(0) - 1
            If A(i) = key Then
                Return i
            End If
        Next
        Return -1
    End Function

End Module
```

顺序查找的时间复杂度是 O(n)。算法的最坏情形要检查每个元素，以判断数组中是否存在要查找的元素。如果数组长度加倍，则算法要执行的比较次数也会加倍。

7.4.2　折半查找

如果查找表中的所有数据元素都按关键字有序顺序组织，则可以采用一种更高效率的

查找方法——折半查找（或称二分查找）。

折半查找的基本思想：由于查找表中的数据元素按关键字有序（假设递增有序），则在查找时可不必逐个顺序比较，而采用跳跃的方式——先与“中间位置”的记录关键字值比较，若相等，则查找成功；若给定值大于“中间位置”的关键字值，则在后半部继续进行折半查找；否则在前半部进行折半查找。

折半查找的过程：先确定待查元素所在区域，然后逐步缩小区域，直到查找成功或失败为止。

假设：待查元素所在区域的下界为 low，上界为 high，则中间位置 mid=(low+high)/2。则折半查找算法的描述如下：

- 设置查找的区间令 low=0，high=A.GetLength(0)–1。
- 计算中间位置 mid=(low+high)/2。
- 若 key=A(mid)，查找成功，返回 mid；若 key＜A(mid)，则令 high=mid–1 后执行上一步骤；若 key＞A(mid)，则令 low=mid+1 后执行上一步骤。
- 若当 low=high 时，key 不等于 A(mid)，则查找失败，返回–1。

由于折半查找要求数据元素的组织方式应具有随机存取的特性，所以，折半查找只适用于以顺序结构组织的有序查找表。折半查找成功的平均查找长度 ASL≈log(n+1)–1。

折半查找的优点是比较次数少，查找速度快。但为了快速查找所付出的代价是要对数据元素按关键字值的大小进行排序，而排序一般是很费时的。所以，折半查找适用于一经建立就很少变动而又经常进行查找的有序表。

例 7-6 在有序数组 A 中折半查找。

程序：

```
Module Module1

    Sub Main()
     Dim A() As Integer = {3, 5, 11, 22, 34, 56, 76, 87, 90, 92, 95, 123, 134}
        Dim k = BinSearch(A, 95)   '搜索 95
        If k >= 0 Then
            Console.WriteLine("查找的数据在数组中的位置为：" + k.ToString())
        Else
            Console.WriteLine("数据没有找到")
        End If
    End Sub

    '该函数在数组中搜索特定的数，如果找到，则返回数据在数组中的位置
    '否则返回-1
    Function BinSearch(ByVal A() As Integer, ByVal key As Integer) As Integer
        Dim low As Integer = 0
        Dim high As Integer = A.GetLength(0) - 1
        While low <= high
            Dim mid = (low + high) / 2
            If key = A(mid) Then
                Return mid     '找到
            Else
```

```
                If key < A(mid) Then
                    high = mid - 1     '在前半部分继续寻找
                Else
                    low = mid + 1      '在后半部分继续寻找
                End If
            End If
        End While
        Return -1     '没有找到
    End Function

End Module
```

7.5　排序算法

排序是计算机内经常进行的一种操作，其目的是将一组“无序”的记录序列调整为“有序”的记录序列。排序可以分为内部排序和外部排序。若整个排序过程不需要访问外存便能完成，则称此类排序问题为内部排序。反之，若参加排序的记录数量很大，整个序列的排序过程不可能在内存中完成，则称此类排序问题为外部排序。内部排序的过程是一个逐步扩大记录的有序序列长度的过程。本节仅讨论内部排序，同时假设要排序的数据均存储在一个一维数组内。

7.5.1　冒泡排序

冒泡排序的基本思想：两两比较待排序记录的关键字，发现两个记录的次序相反时即进行交换，直到没有反序的记录为止。

设想将被排序的数组 R 垂直排列，数组中每个元素 R[i]的值看作是重量为该值的气泡。根据轻气泡不能在重气泡之下的原则，从下往上扫描数组 R。凡扫描到违反本原则的轻气泡，就使其向上“飘浮”（也就是交换轻气泡和重气泡的位置）。如此反复进行，直到最后任何两个气泡都是轻者在上，重者在下为止。此时数组排序完毕。具体步骤如下：

- 假设带排序的数存于数组 R 中（R 的下标范围为 0～n）。
- 第 1 趟扫描：从 R 的结尾处开始，依次比较相邻的两个数值的大小，若发现小者在下、大者在上，则交换二者的位置，即依次比较（R[n]，R[n-1]），（R[n-1]，R[n-2]），…，（R[1]，R[0]）；对于每对气泡（R[j+1]，R[j]），若 R[j+1]<R[j]，则交换 R[j+1]和 R[j]的内容。
- 当第一趟扫描完毕时，最小的数值的就飘浮到该数组的顶部，即最小的数组元素被放在位置 R[0]上。
- 第二趟扫描：类似于第一趟扫描，只不过扫描的范围为 R[1]～R[n]，扫描的结果将使次小的数存放于 R[1]中。
- 最后，经过 n 趟扫描，可以得到排序后的数组 R。

假设数组 R 具有 5 个整数元素分别是 34、12、2、77 和 68，如图 7-6 所示。

0	1	2	3	4
34	12	2	77	68

图 7-6　待排序的数组

第一趟排序扫描的过程如图 7-7 所示。其中，图 7-7（a）中 68＜77 因而 68 和 77 交换位置；图 7-7

（b）中 68＞2 因而不发生交换；图 7-7（c）和图 7-7（d）中 2 被交换到了最前端的位置上。

经过第一趟扫描后，2 将上浮到第 1 位置。

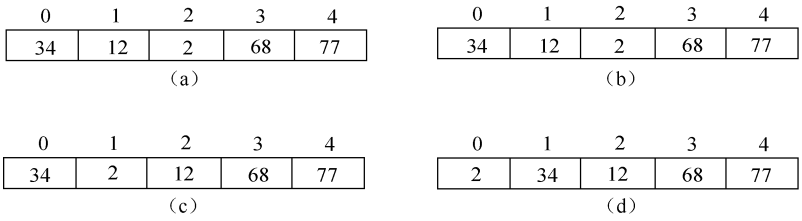


图 7-7　第一趟冒泡排序扫描过程

随后再经过同样的 3 趟扫描，12、34 和 68 将被交换到正确的位置上，排序就完成了。

例 7-7　编写程序，用于整数数组的冒泡排序。

分析：设有大小为 M 的整数数组，编写一个过程，用冒泡排序法对该数组排序，过程参看前面描述的步骤。

程序：

```
    Module Module1

        Sub Main()
          Dim sArray() As Integer = {12, -78, 67, 23, 2, 99, 234, -23, 45, 56,
12, 78}

            '在屏幕上显示数组
            Show(sArray)
            '排序并显示排序后的结果
            Console.WriteLine("数组排序...")
            Bubble(sArray)
            '显示排序后的结果
            Show(sArray)

        End Sub

        '在屏幕上显示数组
        Sub Show(ByVal sArray() As Integer)
            For Each k As Integer In sArray
                Console.Write(k.ToString() + " ")
            Next
            Console.WriteLine()
        End Sub

        '冒泡排序算法
        Sub Bubble(ByVal sArray() As Integer)
            '得到数组的大小
            Dim length As Integer = sArray.GetLength(0)
            For i As Integer = 0 To length - 1
                For j = length - 1 To i + 1 Step -1
```

```
                If sArray(j) < sArray(j - 1) Then
                    '交换
                    Dim temp As Integer = sArray(j)
                    sArray(j) = sArray(j - 1)
                    sArray(j - 1) = temp
                End If
            Next
        Next
    End Sub
End Module
```

从上面的程序可以看到，通过二重循环比较元素之间的大小，若需要排序的元素个数为 n，则比较的次数为

$$(n-1) + (n-2) + \ldots + 1 = n * (n-1) / 2 = (n\text{^}2-n) / 2$$

最差的情况，每次比较都需要交换，其时间复杂度为 O(n^2)。在例 7-7 的冒泡排序上可以有一个小的改进，即当某趟比较如果没有交换发生时，说明排序已经完成（习题 2）。

7.5.2　选择排序

选择排序算法的思想是第一次从数组中选择最小的元素，并将它与第一个元素交换。第二次选择剩余元素中最小的（是所有元素中第二小的），并将其与第二个元素交换。一直这样做下去，直到最后一次选择了第二大的元素，并将它与倒数第二个位置的元素交换（如果有必要的话），使最大的元素位于最后一个位置。经过 i 次选择和交换后，数组中前 i 个元素将按照升序保存在数组的前 i 个位置中。

假设数组 R 具有 5 个整数元素分别是 34、12、2、77 和 68，如图 7-8 所示。

选择排序程序首先判断出最小的元素为 2，位于索引 2 处（即第 3 个元素最小）。于是程序首先将 2 和 34 交换，让确定剩余的元素中最小的 12 处于正确的位置（不用交换），该过程一直继续直到完成整个排序如图 7-9 所示。

0	1	2	3	4
34	12	2	68	77

图 7-8 待排序的数组

	0	1	2	3	4
1	2	12	34	77	68
2	2	12	34	77	68
3	2	12	34	77	68
4	2	12	34	68	77

图 7-9　选择排序的过程

例 7-8　编写程序，用于整数数组的冒泡排序。

分析：设有大小为 M 的整数数组，编写一个过程，用选择排序法对该数组排序，过程参看前面描述的步骤。

程序：

```
Module Module1

    Sub Main()
        Dim sArray() As Integer = {12, -78, 67, 23, 2, 99, 234, -23, 45, 56,
```

```
12, 78}
            '在屏幕上显示数组
            Show(sArray)
            '排序并显示排序后的结果
            Console.WriteLine("数组排序...")
            SelectionSort(sArray)
            '显示排序后的结果
            Show(sArray)

        End Sub

        '在屏幕上显示数组
        Sub Show(ByVal sArray() As Integer)
            For Each k As Integer In sArray
                Console.Write(k.ToString() + " ")
            Next
            Console.WriteLine()
        End Sub

    '选择排序
        Sub SelectionSort(ByVal sArray() As Integer)
            '得到数组的大小
            Dim length As Integer = sArray.GetLength(0)
             '最小一个数的索引
             Dim smallest As Integer
            For i As Integer = 0 To length - 2
                 Smallest = i
                For j As Integer= i+1 To length - 1
                    If sArray(j) < sArray(smallest) Then
                       smallest = j
                    End If
                Next
                  '交换
                  Dim temp As Integer = sArray(smallest)
                  sArray(smallest) = sArray(i)
                  sArray(i) = temp
            Next
        End Sub
```

选择排序的时间复杂度和冒泡排序的时间复杂度是一样的，为 O(n^2)。和冒泡排序一样，选择排序也经过了(n^2–n)/2 次比较。

*7.5.3 快速排序

快速排序（Quicksort）是对冒泡排序的一种改进。由 C、A、R、Hoare 在 1962 年提出。它的基本思想：通过一趟排序将要排序的数据分割成独立的两部分，其中一部分的所有数据都比另外一部分的所有数据都要小，然后再按此方法对这两部分数据分别进行快速排序，整个排序过程可以递归进行，以此达到整个数据变成有序序列。

快速排序的基本算法如下：设要排序的数组是 A(0)…A(N–1)，首先任意选取一个数据（通常选用第一个数据）作为关键数据，然后将所有比它小的数都放到它前面，所有比它大的数都放到它后面，这个过程称为一趟快速排序。

一趟快速排序的算法如下：

（1）设置两个变量 i 和 j，排序开始时 i=0，j=N–1。

（2）以第一个数组元素作为关键数据，赋值给 key，即 key=A(0)。

（3）从 j 开始向前搜索，即由后开始向前搜索（j=j–1），找到第一个小于 key 的值 A(j)，令 A(i)=A(j)。

（4）从 i 开始向后搜索，即由前开始向后搜索（i=i+1），找到第一个大于 key 的 A(i)，令 A(j)=A(i)。

（5）重复第 3、4、5 步，直到 i=j。

（6）令 A(i)=key。

设有数组 A()={50, 39, 64, 90, 72, 12, 29}，一趟快速排序后数组的交换过程如图 7-10 所示。

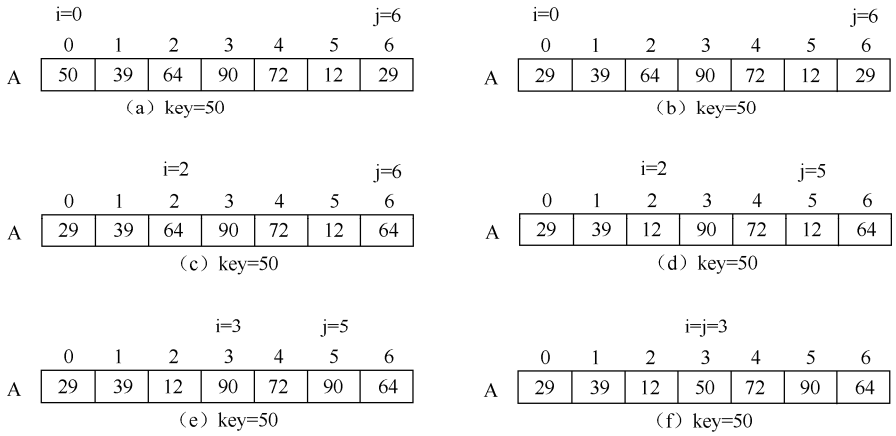


图 7-10　一趟快速排序的交换过程

图 7-10（a）是初始状态，在图 7-10（b）中，由于 A(6)＜key，因而被放置到 A(0)。在图 7-10（c）中，当 i=1 时，没有改变，直到 i=2，A(2)的值放置到 A(6)。图 7-10（d）和图 7-10（e）重复这一过程，直到图 7-10（f），此时 i=j=3，将 key 的值放入。经过这样的一趟排序后，凡是比 key=50 大的值都移动到了数组的后半部分，比 50 小的都在前面。

要完成整个数组的排序，只需要递归调用此过程，以 50 为中点分割这个数据序列，分别对前面一部分和后面一部分进行类似的快速排序，从而完成全部数据序列的快速排序，也就是对 A(0)～A(2)排序，对 A(4)～A(6)排序，最后把此数据序列变成一个有序的序列。

例 7-9　对整数数组给出快速排序的递归算法。

分析：只需要按照前面的分析，写出程序即可。

程序：

```
Module Module1

    Sub Main()
```

```
            Dim A() As Integer = {50, 39, 64, 90, 72, 12, 29}
            Console.WriteLine("排序前：")
            '显示 A 数组
            Show(A)

            '排序
            QkSort(A, 0, A.GetLength(0) - 1)

            Console.WriteLine("排序后：")
            Show(A)
        End Sub

        '显示数组的 A 数据
        Sub Show(ByVal A() As Integer)
            For Each k As Integer In A
                Console.Write(k.ToString() + " ")
            Next
            Console.WriteLine()
        End Sub

        ' 快速排序
        'A 是待排序的数组
        'i 和 j 指示了对数组从 i 到 j 处的数据进行排序，i<j
        Sub QkSort(ByVal A() As Integer, ByVal i As Integer, ByVal j As Integer)
            If i < j Then
                '对数组 A 调用 QkPass 函数进行一趟快速排序
                'i 和 j 指示了排序的起始和终了位置（下标）
                '返回值指示了一趟排序后的分割点
                Dim k = QkPass(A, i, j)
                '对前一部分继续快速排序，递归调用
                QkSort(A, i, k - 1)
                '对后一部分快速排序
                QkSort(A, k + 1, j)
            End If
        End Sub

        '一趟快速排序的函数，对数组 A 从 i 到 j 快速排序，并返回分割点
        Function QkPass(ByVal A() As Integer, ByVal i As Integer, ByVal j As Integer)
As Integer
            '存储关键字
            Dim key As Integer = A(i)
            While i < j
                While i < j And A(j) >= key
                    j = j - 1          '从后向前搜寻比 key 小的值
                End While
                A(i) = A(j)   '找到后放入 A(i)
                While i < j And A(i) <= key
```

```
                i = i + 1       '从前向后搜寻比 key 大的值
            End While
            A(j) = A(i)    '找到后放入 A(j)
        End While

        '循环结束时，i=j，放入 key 值，并返回 i
        A(i) = key
        Return i
    End Function
End Module
```

快速排序每次将待排序数组分为两个部分，在理想状况下，每一次都将待排序数组划分成等长两个部分，则需要 $\log_2 n$ 次划分。而在最坏情况下，即数组已经有序或大致有序的情况下，每次划分只能减少一个元素，快速排序将退化为冒泡排序，所以，快速排序时间复杂度下界为 O(nlogn)，最坏情况为 O(n^2)。在实际应用中，快速排序的平均时间复杂度为 O(nlogn)。

快速排序的最坏情况基于每次划分对关键数据的选择。基本的快速排序选取第一个元素作为关键数据。这样在数组已经有序的情况下，每次划分将得到最坏的结果。一种比较常见的优化方法是随机化算法，即随机选取一个元素作为关键数据。这种情况下虽然最坏情况仍然是 O(n^2)，但最坏情况不再依赖于输入数据，而是由于随机函数取值不佳。实际上，随机化快速排序得到理论最坏情况的可能性仅为 1/(2^n)。所以，随机化快速排序可以对于绝大多数输入数据达到 O(nlogn)的期望时间复杂度。

*7.6　常用算法简介

对于计算机科学来说，算法（Algorithm）的概念是至关重要的。例如，在一个大型软件系统的开发中，设计出有效的算法将起决定性的作用。通俗地讲，算法是指解决问题的一种方法或一个过程。程序（Program）与算法不同。程序是算法用某种程序设计语言的具体实现。

7.6.1　递归与分治

任何一个可以用计算机求解的问题所需的计算时间都与其规模有关。问题的规模越小，解题所需的计算时间往往也越少，从而也较容易处理。例如，对于 n 个元素的排序问题，当 n=1 时，不需任何计算。n = 2 时，只要作一次比较即可排好序。n = 3 时只要作两次比较即可。而当 n 较大时，问题就不那么容易处理了。要想直接解决一个较大的问题，有时是相当困难的。分治法的设计思想是，将一个难以直接解决的大问题，分割成一些规模较小的相同的问题，各个击破，分而治之。如果原问题可分割成 k 个子问题，1<k<=n，且这些子问题都可解，利用这些子问题的解求出原问题的解，那么这种分治法就是可行的。由分治法产生的问题往往是原问题的较小模式，这就为使用递归技术提供了方便。在这种情况下，反复应用分治手段，可以使子问题与原问题类型一致而其规模却不断缩小，最终使子问题缩小到很容易求解的规模。这样，就自然导致递归算法的产生。

一个直接或间接的调用自身的算法称为递归算法，一个使用函数自身给出定义的函数

成为递归函数。使用递归往往使函数的定义和算法的描述简洁。关于 Visual Basic 递归函数的调用在 5.6.9 节已经讲述过了。

分治的基本思想：对于一个规模为 n 的问题，若该问题可以容易地解决（如规模 n 较小）则直接解决，否则将其分解为 k 个规模较小的子问题，这些子问题互相独立且与原问题形式相同，递归地解这些子问题，然后将各子问题的解合并得到原问题的解。

分治法所能解决的问题一般具有以下几个特征：

- 该问题的规模缩小到一定的程度就可以容易地解决。
- 该问题可以分解为若干个规模较小的相同问题。
- 利用该问题分解出的子问题的解可以合并为该问题的解。
- 该问题所分解出的各个子问题是相互独立的，即子问题之间不包含公共的子问题。

上述的第一条特征是绝大多数问题都可以满足的，因为问题的计算复杂性一般是随着问题规模的增加而增加；第二条特征是应用分治法的前提，它也是大多数问题可以满足的，此特征反映了递归思想的应用；第三条特征是关键，能否利用分治法完全取决于问题是否具有第三条特征，如果具备了第一条和第二条特征，而不具备第三条特征，则可以考虑用贪心法或动态规划法；第四条特征涉及到分治法的效率，如果各子问题是不独立的则分治法要做许多不必要的工作，重复地解公共的子问题，此时虽然可用分治法，但一般用动态规划法较好。

根据分治法的分割原则，原问题应该分为多少个子问题才较适宜？各个子问题的规模应该怎样才较适当？这些问题没有确定的答案。但人们从大量实践中发现，在用分治法设计算法时，最好使子问题的规模大致相同。换句话说，将一个问题分成大小相等的 k 个子问题的处理方法是行之有效的。许多问题可以取 k = 2。这种使子问题规模大致相等的做法是出自一种平衡（Balancing）子问题的思想，它几乎总是比子问题规模不等的做法要好。

折半查找则是分治策略的典型的例子。折半查找的过程 BinSearch 中的 While 循环是决定算法快慢的关键。容易看出，每执行一次算法的 While 循环，待搜索数组的大小减少 1/2。因此，在最坏的情况下，While 循环被执行了 O(logn)次。循环体内运算需要 O(1)时间，因此，整个算法在最坏的情况下的计算时间复杂度为 O(logn)。

快速排序则是基于分治策略的排序算法。回顾快速排序的过程，不难看出快速排序是按分治法的三个步骤：分解，递归求解与合并来完成的。快速排序的运行时间按与划分是否对称有关。最坏的情况发生在划分过程产生的两个区域分别包含 n–1 个元素和 1 个元素的时候。此时其时间复杂度为 $O(n^2)$。在最好的情况下，每次划分所取得基准都恰好是中值，此时的时间复杂度为 O(nlogn)。

7.6.2 动态规划

动态规划（Dynamic Programming）是运筹学的一个分支，是求解决策过程（Decision Process）最优化的数学方法。20 世纪 50 年代初美国数学家 R・E・Bellman 等人在研究多阶段决策过程（Multistep Decision Process）的优化问题时，提出了著名的最优化原理（Principle Of Optimality），把多阶段过程转化为一系列单阶段问题，利用各阶段之间的关系，逐个求解，创立了解决这类过程优化问题的新方法——动态规划。1957 年出版了他的名著《Dynamic Programming》，这是该领域的第一本著作。

动态规划问世以来，在经济管理、生产调度、工程技术和最优控制等方面得到了广泛的应用。如最短路线、库存管理、资源分配、设备更新、排序、装载等问题，用动态规划方法比用其他方法求解更为方便。

动态规划程序设计是对解最优化问题的一种途径、一种方法，而不是一种特殊算法。不象前面所述的那些搜索或数值计算那样，具有一个标准的数学表达式和明确清晰的解题方法。动态规划程序设计往往是针对一种最优化问题，由于各种问题的性质不同，确定最优解的条件也互不相同，因而动态规划的设计方法对不同的问题，有各具特色的解题方法，而不存在一种万能的动态规划算法，可以解决各类最优化问题。因此，读者在学习时，除了要对基本概念和方法正确理解外，必须具体问题具体分析处理，以丰富的想象力去建立模型，用创造性的技巧去求解。我们也可以通过对若干有代表性的问题的动态规划算法进行分析、讨论，逐渐学会并掌握这一设计方法。

动态规划算法通常用于求解具有某种最优性质的问题。在这类问题中，可能会有许多可行解。每一个解都对应于一个值，我们希望找到具有最优值的解。动态规划算法与分治法类似，其基本思想也是将待求解问题分解成若干个子问题，先求解子问题，然后从这些子问题的解得到原问题的解。与分治法不同的是，适合用动态规划求解的问题，经分解得到子问题往往不是互相独立的。若用分治法来解这类问题，则分解得到的子问题数目太多，有些子问题被重复计算了很多次。如果我们能够保存已解决的子问题的答案，而在需要时再找出已求得的答案，这样就可以避免大量的重复计算，节省时间。我们可以用一个表来记录所有已解的子问题的答案。不管该子问题以后是否被用到，只要它被计算过，就将其结果填入表中。这就是动态规划法的基本思路。具体的动态规划算法多种多样，但它们具有相同的填表格式。

任何思想方法都有一定的局限性，超出了特定条件，它就失去了作用。同样，动态规划也并不是万能的。适用动态规划的问题必须满足最优化原理和无后效性。

最优化原理也就是最优子结构性质：一个最优化策略具有这样的性质，不论过去状态和决策如何，对前面的决策所形成的状态而言，余下的诸决策必须构成最优策略。简而言之，一个最优化策略的子策略总是最优的。一个问题满足最优化原理又称其具有最优子结构性质。

无后效性是指将各阶段按照一定的次序排列好之后，对于某个给定的阶段状态，它以前各阶段的状态无法直接影响它未来的决策，而只能通过当前的这个状态。换句话说，每个状态都是过去历史的一个完整总结。这就是无后向性，又称为无后效性。

子问题的重叠性。动态规划将原来具有指数级复杂度的搜索算法改进成了具有多项式时间的算法。其中的关键在于解决冗余，这是动态规划算法的根本目的。动态规划实质上是一种以空间换时间的技术，它在实现的过程中，不得不存储产生过程中的各种状态，所以，它的空间复杂度要大于其他的算法。

例 7-10　0-1 背包问题。给定 n 种物品和一背包。物品 i 的重量是 wi，其价值为 vi，背包的容量为 c。问应如何选择装入背包中的物品，使得装入背包中物品的总价值最大？在选择装入背包的物品时，对每种物品 i 只有两种选择，即装入背包或不装入背包。不能将物品 i 装入背包多次，也不能只装入部分的物品 i。因此，该问题称为 0-1 背包问题。

分析：首先，该问题具有最优子结构。也就是说，假设 y_1，y_2，…，y_n 是装入背包的

一种最优方案。那么，y_1，y_2，…，y_{n-1} 也是装入到容量为 c-wn 背包中的一种最优方案。其次，不难看出，该问题具有递归的性质。

假设函数 f(n,c)表示将 n 件物品装入背包容量为 c 可获得的最大价值，则对于第一件物品而言，具有两种选择：装入或者不装入。若选择装入，则剩余 n-1 件物品装入容量为 c-w1 的背包中，构成最优子结构的解。反之，则将 n-1 件物品装入容量为 c 背包中。显然，可得

$$f(n,c)=MAX(f(n-1,c),f(n-1,c-w1)+v1)$$

递归求解，即可以获得最优解。

而递归的边界条件是，若只有一件物品，当背包还有容量时，则装入，否则不装入。

然而，这样做的效率是低下的。动态规划的另一个特点就是记录曾经计算过的值。这样，在以后需要的时候，不需要重新计算，而是可以直接得到。在此我们使用二维数组 m 来记录，m(i,j)表示将第 i，i+1，…，n 个物品装入到容量为 j 的背包中，可获得的最大价值。这样，求解背包问题，则是对 m 数组的填充。

根据以上的分析，假设有如下过程。

程序：

```
    Sub KnapSack(ByVal v() As Integer, ByVal w() As Integer, ByVal c As Integer,
ByVal n As Integer, ByVal m(,) As Integer)
    '程序的主要功能是对 m 数组的填充
    'm(i,j)表示将第 i，i+1，… ，n 个物品装入到容量为 j 的背包中，可获得的最大价值
    '当 i=n 时，只有一件物品有可能放入：

        Dim jMax = Math.Min(w(n - 1) - 1, c)
        For j As Integer = 0 To jMax
            m(n, j) = 0
        Next
        For j As Integer = w(n - 1) To c
            m(n, j) = v(n - 1)
        Next

        '自底向上推演，计算 n-1 到 2 的情形：
        For i As Integer = n - 1 To 2 Step -1
            jMax = Math.Min(w(n - 1) - 1, c)
            For j As Integer = 0 To jMax
                m(i, j) = m(i + 1, j)
            Next
            For j As Integer = w(i - 1) To c
                m(i, j) = Math.Max(m(i + 1, j), m(i + 1, j - w(i - 1)) + v(i - 1))
            Next
        Next

        '考虑第一件物品选取或不选取的情形：
        m(1, c) = m(2, c)
        If c >= w(0) Then
            m(1, c) = Math.Max(m(1, c), m(2, c - w(0)) + v(0))
        End If
    End Sub
```

注意的是该过程计算后，在 m(1,c)中的值就是背包问题的最优值。具体该选取那些物品其实也记录在 m 数组中了，可以用过程 TraceBack 构造如下。如果 m(1,c)=m(2,c)，则第一件物品不选取，x1=0，否则 x1=1。当 x1=0 时，由 m(2,c)继续构造最优解，当 x1=1 时，由 m(2,c–w0)继续构造最优解：

```
    Sub TraceBack(ByVal m(,) As Integer, ByVal w() As Integer, ByVal c As Integer,
ByVal n As Integer, ByVal x() As Integer)
          For i As Integer = 1 To n - 1
             If m(i, c) = m(i + 1, c) Then
                x(i) = 0
             Else
                x(i) = 1
                c = c - w(i - 1)
             End If

             If m(n, c) > 0 Then
                x(n) = 1
             Else
                x(n) = 0
             End If
          Next
       End Sub
```

7.6.3　贪心算法

贪心算法（又称为贪婪算法）是指，在对问题求解时，总是做出在当前看来是最好的选择。也就是说，不从整体最优上加以考虑，他所做出的仅是在某种意义上的局部最优解。贪心算法不是对所有问题都能得到整体最优解，但对范围相当广泛的许多问题他能产生整体最优解或者是整体最优解的近似解。

当一个问题具有最优子结构性质时，我们会想到用动态规划法去解它。但有时会有更简便的算法。我们来看一个找硬币的例子。假设有四种硬币，它们的面值分别为二角五分、一角、五分和一分。现在要找给某顾客六角三分钱。这时，我们会不假思索地拿出 2 个二角五分硬币，1 个一角的硬币和 3 个一分的硬币交给顾客。这种找硬币方法与其他的找法相比，拿出的硬币个数是最少的。这里，使用了这样的找硬币算法：首先选出一个面值不超过六角三分的最大硬币，即二角五分；然后从六角三分中减去二角五分，剩下三角八分；再拿出一个面值不超过三角八分的最大硬币，即又一个二角五分，如此一直做下去。这个找硬币方法实际上就是贪心算法。顾名思义，贪心算法总是作出在当前看来是最好的选择。也就是说贪心算法并不从整体最优上加以考虑，它所作出的选择只是在某种意义上的局部最优。当然，我们希望贪心算法得到的最终结果也是整体最优的。上面所说的找硬币算法得到的结果就是一个整体最优解。找硬币问题本身具有最优子结构性质，它可以用动态规划算求解。但我们看到，用贪心算法更简单，更直接且解题效率更高。这利用了问题本身的一些特性。例如，上述找硬币的算法利用了硬币面值的特殊性。如果硬币的面值改为一分、五分和一角一分 3 种，而要找给顾客的是一角五分钱。还用贪心算法，我们将找给顾客 1 个一角一分硬币和 4 个一分的硬币。然而 3 个五分的硬币显然是最好的找法。虽然贪心算法不是对所有题都能得

到整体最优解，但对范围相当广的许多问题它能产生整体最优解。

贪心算法通过一系列的选择来得到一个问题的解。它所作的每一个选择都是当前状态下某种意义的最好选择，即贪心选择。希望通过每次所作的贪心选择导致最终结果是问题的一个最优解。这种启发式的策略并不总能奏效，然而在许多情况下确能达到预期的目的。

对于一个具体的问题，我们怎么知道是否可用贪心算法来解此问题，以及能否得到问题的一个最优解呢？这个问题很难给予肯定的回答。但是，从许多可以用贪心算法求解的问题中可以看到它们一般具有两个重要的性质：贪心选择性质和最优子结构性质。

1．贪心选择性质

贪心选择性质是指所求问题的整体最优解可以通过一系列局部最优的选择，即贪心选择来达到。这是贪心算法可行的第一个基本要素，也是贪心算法与动态规划算法的主要区别。在动态规划算法中，每步所作的选择往往依赖于相关子问题的解。因而只有在解出相关子问题后，才能作出选择。而在贪心算法中，仅在当前状态下作出最好选择，即局部最优选择。然后再去解作出这个选择后产生的相应的子问题。贪心算法所作的贪心选择可以依赖于以往所作的选择，但决不依赖于将来所作的选择，也不依赖于子问题的解。正是由于这种差别，动态规划算法通常以自底向上的方式解各子问题，而贪心算法则通常以自顶向下的方式进行。以迭代的方式作出相继的贪心选择，每作一次贪心选择就将所求问题简化为一个规模更小的子问题。

对于一个具体问题，要确定它是否具有贪心选择性质，我们必须证明每一步所作的贪心选择最终导致问题的一个整体最优解。首先考察问题的一个整体最优解，并证明可修改这个最优解，使其以贪心选择开始。而且作了贪心选择后，原问题简化为一个规模更小的类似子问题。然后，用数学归纳法证明，通过每一步作贪心选择，最终可得到问题的一个整体最优解。其中，证明贪心选择后的问题简化为规模更小的类似子问题的关键在于利用该问题的最优子结构性质。

2．最优子结构性质

当一个问题的最优解包含着它的子问题的最优解时，称此问题具有最优子结构性质。问题所具有的这个性质是该问题可用动态规划算法或贪心算法求解的一个关键特征。

最后，回顾 6.4.2 节，带权图的最短路径问题（例 6-13）。例 6-13 的 Dijkstra 算法是应用贪心算法的一个典型的例子。其贪心选择是从 T 集合中选择具有最短路径的顶点 u，从而确定从源（也就是起点，该问题又称为单源最短路径问题）到 u 的最短路径的长度 d(u)。这也就是该问题具有贪心性质。

其次，该问题还具有最优子结构性质。关于这一点的证明，可参阅其他书籍。

7.6.4 回溯法

回溯法有“通用的解题法”之称。用它可以系统地搜索一个问题的所有解或任一解。回溯法是一个既带有系统性又带有跳跃性的搜索算法。它在包含问题的所有解的解空间树中，按照深度优先的策略，从根结点出发搜索解空间树。算法搜索至解空间树的任一结点时，总是先判断该结点是否肯定不包含问题的解。如果肯定不包含，则跳过以该结点为根

的子树的系统搜索，逐层向其祖先结点回溯。否则，进入该子树，继续按深度优先的策略进行搜索。回溯法在用来求问题的所有解时，要回溯到根，且根结点的所有子树都已被搜索遍才结束。而回溯法在用来求问题的任一解时，只要搜索到问题的一个解就可结束。这种以深度优先的方式系统地搜索问题的解的算法称为回溯法，它适用于解一些组合数较大的问题。

应用回溯法解问题时，首先应明确定义问题的解空间。问题的解空间应至少包含问题的一个（最优）解。例如，对于有 n 种可选择物品的 0-1 背包问题，其解空间由长度为 n 的 0-1 矢量构成。该解空间包含了对变量的所有可能的 0-1 赋值。当 n=3 时，其解空间为

{(0,0,0),(0,1,0),(0,0,1),(1,0,0),(0,1,1),(1,0,1),(1,1,0),(1,1,1)}

定义了问题的解空间后，还应将解空间很好地组织起来，使得用回溯法能方便地搜索整个解空间。通常将解空间组织成树或图的形式。

例如，对于 n=3 时的 0-1 背包问题，其解空间用一棵完全二叉树表示，如图 7-11 所示。

解空间树的第 i 层到第 i+1 层边上的标号给出了变量的值。从树根到叶的任一条路径表示解空间的一个元素。例如，从根结点到结点 H 的路径相应于解空间的元素(1,1,1)。

确定了解空间的组织结构后，回溯法就从开始结点（根结点）出发，以深度优先的方式搜索整个解空间。这个开始结点就成为一个活结点，同时也成为当前的扩展结点。在当前的扩展结点处，搜索向纵深方向移至一个新结点。这个新结点就成为一个新的活结点，并成为当前扩展结点。如果在当前的扩展结点处不能再向纵深方向移动，则当前的扩展结点就成为死结点。换句话说，这个结点不再是一个活结点。此时，应往回移动（回溯）至最近的一个活结点处，并使这个活结点成为当前的扩展结点。回溯法即以这种工作方式递归地在解空间中搜索，直至找到所要求的解或解空间中已无活结点时为止。

例如，对于 n=3 时的 0-1 背包问题，考虑下面的具体实例：w=[16,15,15]，p = [45,25,25]，c=30。我们从图 7-11 的根结点开始搜索其解空间。开始时根结点是唯一的活结点，也是当前的扩展结点。在这个扩展结点处，可以沿纵深方向移至结点 B 或结点 C。假设选择先移至结点 B。此时，结点 A 和结点 B 是活结点，结点 B 成为当前扩展结点。由于选取了 w1，故在结点 B 处剩余背包容量 r=14，获取的价值为 45。从结点 B 处，可以移至结点 D 或 E，由于移至结点 D 至少需要 w2=15 的背包容量，而我们现在仅有的背包容量是 14，故移至结点 D 导致一个不可行解。而搜索至结点 E 不需要背包容量，因而是可行的。从而选择移至结点 E。此时，E 成为新的扩展结点，结点 A、B 和 E 是活结点。在结点 E 处，r=14，获取的价值为 45。从结点 E 处，可以向纵深移至结点 J 或 K。移至结点 J 导致一个不可行解，而移向结点 K 是可行的，于是移向结点 K，它成为一个新的扩展结点。由于结点 K 是一个叶结点，故我们得到一个可行解。这个解相应的价值为 45。xi 的取值由根结点到叶结点 K 的路径所唯一确定，即 x=(1,0,0)。由于在结点 K 处已不能再向纵深扩展，所以结点 K 成为死结点。于是返回到结点 E 处。此时在结点 E 处也没有可扩展的结点，它也成为死结点。

接下来又返回到结点 B 处。结点 B 同样也成为死结点，从而结点 A 再次成为当前扩展结点。结点 A 还可继续扩展，从而到达结点 C。此时，r=30，获取的价值为 0。从结点 C 可移向结点 F 或 G。假设我们移至结点 F，它成为新的扩展结点。结点 A、C 和 F 是活结点。在结点 F 处，r= 15，获取的价值为 25。从结点 F，向纵深移至结点 L 处，此时，r=0，

获取价值为 50。由于 L 是一个叶结点，而且是迄今为止找到的获取价值最高的可行解，因此，记录这个可行解。结点 L 不可扩展，我们又返回到结点 F 处。按此方式继续搜索，可搜索遍整个解空间。搜索结束后找到的最好解是相应 0-1 背包问题的最优解。

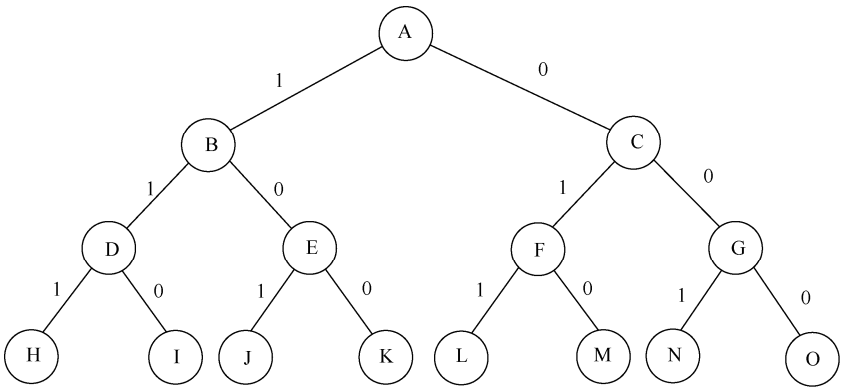


图 7-11　0-1 背包问题的解空间树

习题

1．编写程序，使用冒泡排序对 10 个整数排序。

2．改进冒泡排序程序，使其当数据已经有序时，直接排序的过程。

3．编写程序，使用快速排序对 10 个整数排序。

4．改进书中的快速排序程序，随即选取关键数据。

5．*编写程序，使用冒泡排序对电话号码簿按人名的字典顺序排序。

6．使用顺序查找，对第 5 题的数据查找一个人名是否在电话簿中。

7．使用折半查找，对排序后的电话簿（第 5 题）进行查找。

8．给定 K 个整数的序列{N_1、N_2、...、N_K}，其任意连续子序列可表示为{N_i,N_{i+1},…,N_j}，其中 1≤i≤j≤K。最大连续子序列是所有连续子序列中元素和最大的一个，例如，给定序列{–2，11，–4，13，–5，–2}，其最大连续子序列为{11，–4，13}，最大和为 20。

9．数论中有许多猜想尚未解决，其中有一个被称为“角谷猜想”的问题，这个问题是这样描述的：任何一个大于 1 的自然数，如果是奇数，则乘以 3 再加 1；如果是偶数，则除以 2；得出的结果继续按照前面的规则进行运算，最后必定得到 1。现在请你编写一个程序验证。

10．某部队进行新兵队列训练，将新兵从 1 开始按顺序依次编号，并排成一行横队，训练的规则如下：从头开始 1 至 2 报数，凡报到 2 的出列，剩下的向小序号方向靠拢，再从头开始进行 1 至 3 报数，凡报到 3 的出列，剩下的向小序号方向靠拢，继续从头开始进行 1 至 2 报数，以后从头开始轮流进行 1 至 2 报数、1 至 3 报数直到剩下的人数不超过三人为止。编写程序，输入数 N 为最开始的新兵人数（20＜N＜6000），输出剩下的新兵最初的编号。

11．在医院打点滴（吊针）时，如果滴起来有规律，先是滴一滴，停一下；然后滴二滴，停一下；再滴三滴，停一下，……，现在有一个问题：这瓶盐水一共有 v 毫升，每一滴是 d 毫升，每一滴的速度是一秒（假设最后一滴不到 D 毫升，则花费的时间也算一秒），停一下的时间也是一秒，这瓶水什么时候能滴完呢（0＜d＜v＜1000）？

*第 8 章　综合案例设计

引言

本章是对前几章的进一步引申。补充了 Windows 环境下编程的基本知识。讲述了窗体和消息驱动的基本概念，以及网络编程和数据库编程的初步介绍。

教学目的

- 初步了解 Windows 环境下的编程。
- 理解 Windows 的消息机制。
- 初步了解网络编程。
- 初步了解数据库编程。

8.1　Windows 环境下编程简介

Windows 编程使用事件驱动的程序设计思想。是一个“基于事件的，消息驱动的”操作系统。在事件驱动的程序结构中，程序的控制流程不再由事件的预定发生顺序来决定，而是由实际运行时各种事件的实际发生来触发，而事件的发生可能是随机的、不确定的，并没有预定的顺序。事件驱动的程序允许用户用各种合理的顺序来安排程序的流程。事件驱动是一种面向用户的程序设计方法，在程序设计过程中除了完成所需要的程序功能之外，更多地考虑了用户可能的各种输入（消息），并有针对性地设计相应的处理程序。事件驱动程序设计也是一种“被动”式的程序设计方法，程序开始运行时，处于等待消息状态，然后取得消息并对其作出相应反应，处理完毕后又返回处于等待消息的状态。

8.1.1　Windows 的消息机制

窗口是 Windows 本身及 Windows 环境下的应用程序的基本界面单位，但是很多人都误以为只有具有标题栏、状态栏、最大化、最小化按钮这样标准的方框才称为窗口。其实窗口的概念很广，如按钮和对话框等也是窗口，只不过是一种特殊的窗口罢了。

从用户的角度看，窗口就是显示在屏幕上的一个矩形区域，其外观独立于应用程序，事实上它就是生成该窗口的应用程序与用户间的直观接口；从应用程序的角度看，窗口是受其控制的一部分矩形屏幕区。应用程序生成并控制与窗口有关的一切内容，包括窗口的大小、风格、位置及窗口内显示的内容等。用户打开一个应用程序后，程序将创建一个窗口，并在那里默默地等待用户的要求。每当用户选择窗口中的选项，程序即对此做出响应。

在 Windows 下执行一个程序，只要用户进行了影响窗口的动作（如改变窗口大小或移

动、单击鼠标等）该动作就会触发一个相应的“事件”。系统每次检测到一个事件时，就会给程序发送一个“消息”，从而使程序可以处理该事件。每个 Windows 应用程序都是基于事件和消息，而且包含一个主事件循环，它不停地、反复地检测是否有用户事件发生。每次检测到一个用户事件，程序就对该事件做出响应，处理完再等待下一个事件的发生。Windows 下的应用程序不断地重复这一过程，直至用户终止程序。

例 8-1 Hello World 程序。为了对 Visual Basic 图形界面编程有一个初步的认识，现在来创建一个简单的 Visual Basic 程序。程序启动后显示一个带有按钮的窗体，单击“Say Hello”按钮后，显示一个有“Hello, world”字样的对话框，如图 8-1 所示。

（a）　　（b）

图 8-1　Hello World 程序

步骤：

（1）启动 Visual Studio 2008 从文件菜单中选择“新建”，再选择“项目”，在弹出的对话框的左边“项目类型”下选择 Visual Basic 下的 Windows，接着在右边“模板”中选择“Windows 窗体应用程序”。如图 8-2 所示，将名称一栏改为“HelloWorld”后，单击“确定”按钮。

图 8-2　新建 Visual Basic 项目

（2）Visual Studio 随后生成项目，将看到如图 8-3 的界面。也许看到的界面和这里的有一些不同之处，但这无关紧要。

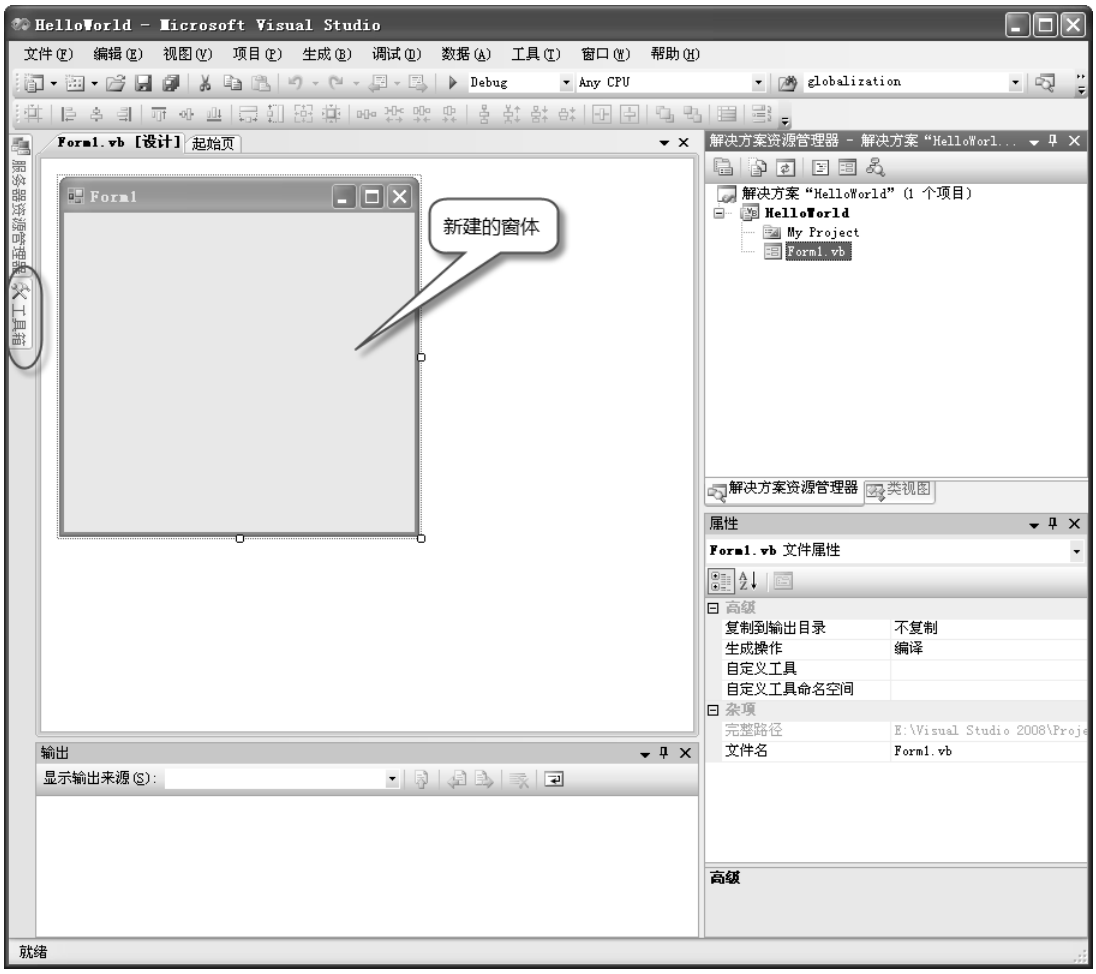


图 8-3　Visual Basic 新生成项目的界面

（3）将鼠标指向窗口边的工具箱，在弹出的窗口展开公共控件，如图 8-4 所示。选择 Button 项，随后在 Form1 窗体上单击一下，于是在 Form1 的窗体上出现一个按钮，名为“Button1”，如图 8-5 所示。

（4）在属性窗口，如图 8-6 所示（如果属性窗口没有显示出来，可以单击 Button1 按钮，在弹出的菜单中选择“属性”），找到 Text 行，将 Button1 改为 Say Hello。这时，再单击 Form1 窗口，可以看到按钮上的字符 Button1 变为了 Say Hello。

（5）再在工具箱上选择 Label 控件，将它放置到按钮的上面，用同样的方法将 Label 控件属性中的 Text 项改为“欢迎到来”（没有引号）。

（6）到目前为止，已经做完了界面设计。在做这一步之前，保存目前的成果是一个好的习惯。要保存成果，可以在文件菜单中选择全部保存。下面进行实际的程序编写，使得程序运行后，单击按钮能够显示 Hello，world 对话框。在按钮上双击，Visual Basic 将打开程序窗口。此时，光标已经停在了将要编码的地方——End Sub 一行的上方，如果不是，

请将光标移到这一行。在光标处键入如下语句，如图 8-7 所示。

图 8-4　Visual Basic 工具箱窗口

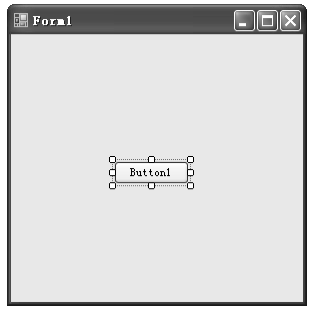


图 8-5　含有一个按钮的 Form1 窗口

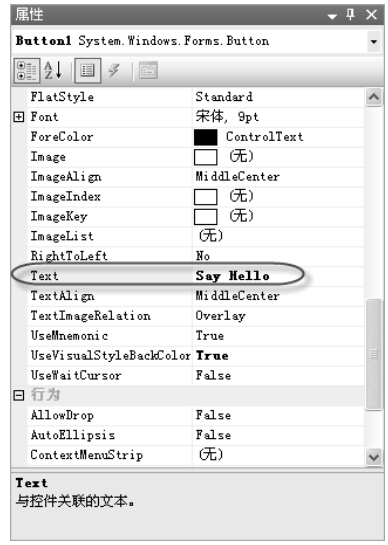


图 8-6　属性窗口

```
    Private Sub Button1_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Button1.Click
        MessageBox.Show("Hello, world!")
    End Sub
```

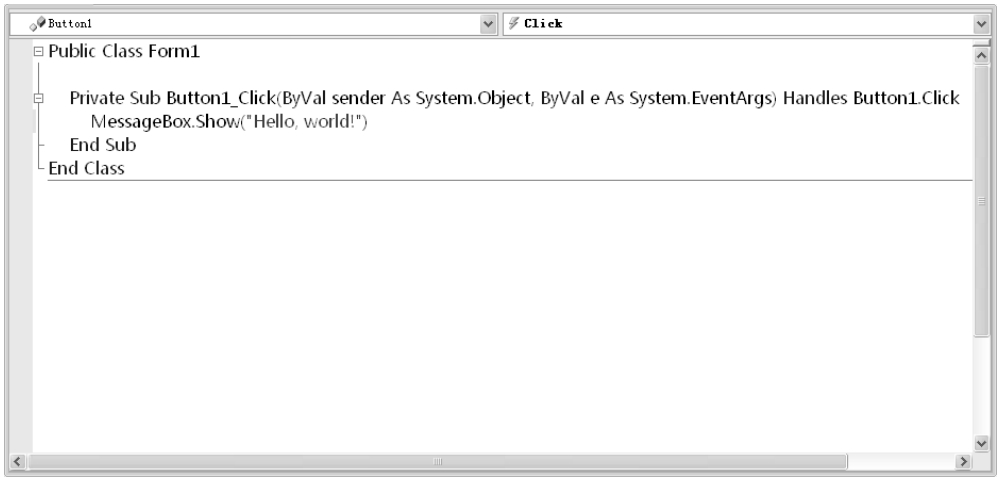


图 8-7　Visual Basic 程序窗口

（7）至此，程序已全部编写完毕。接下来就可以运行了，在此之前，再次保存成果。保存完毕后，从“生成”菜单下选择生成 Hello World，如果出现错误，请回顾前面的步骤。生成成功后，在“调试”菜单中选择开始执行，程序开始运行，出现图 8-1 所示的窗口，单击 Say Hello 按钮，将会看到 Hello World 的对话框的出现。

前面已经讲过，Windows 是通过事件对对象进行驱动的。事实上，多数程序都是事件驱动的，即执行流程是由外界发生的事件所确定的。事件是一个信号，它告知应用程序有重要情况发生。例如，用户单击窗体上的某个控件时，窗体引发一个 Click 事件并调用一个处理该事件的过程。事件还允许在不同任务之间进行通信。例如，应用程序脱离主程序执行一个排序任务。若用户取消这一排序，应用程序可以发送一个取消事件让排序过程停止。

例8-2　创建含有两个文本框的例子，当在第一个文本框中键入字符时，第二个文本框的内容始终和第一个文本框的内容保持一致。

分析：当在一个文本框中键入字符时，会触发事件——TextChanged。为该事件编写代码，将第一个文本框中的内容复制到第二个文本框中。这样就达到了目的。

步骤：

（1）开始一个新的 Windows 项目，命名为 StringCopy，在 Form1 窗体上放置两个文本框控件，同时放置两个 Label 标签标识这两个文本框，并设置属性如表 8-1～表 8-4 所示。

表 8-1　Label1 控件的属性

属性	值
Name	LblInput
Text	输入的字符

表 8-2　Label2 控件的属性

属性	值
Name	LblCopy
Text	复制的字符

表 8-3　TextBox1 控件的属性

属性	值
Name	TxtInput

表 8-4　TextBox2 控件的属性

属性	值
Name	TxtCopy
ReadOnly	True

（2）调整好控件的大小和位置后，窗体看起来如图 8-8 所示。

图 8-8　StringCopy 程序的界面

（3）在 Form1 窗体上的任一处单击，从弹出的菜单中选择查看代码，Visual Basic 2008 打开源代码窗口，在源代码的窗口顶部，有两个下拉框，如图 8-9 所示。

图 8-9　事件与属性的选择

单击左边一个下拉框的箭头，可以看到里面含有窗体上放置的所有控件的名称，如图 8-10 所示。

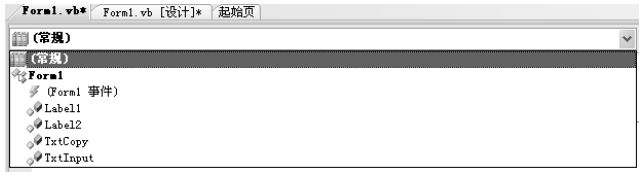


图 8-10　选择接受事件的控件

选择 txtInput 控件后，在同样单击右边的下拉框箭头，可以看到在这里边列出了所有 Windows 已经预先定义好了的，txtInput 控件可以处理的事件，如图 8-11 所示。

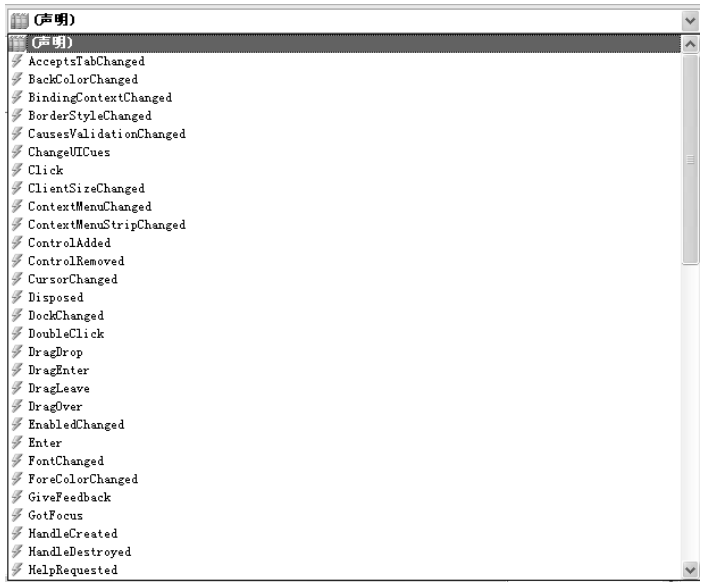


图 8-11　控件的事件列表

拉动右边的滑块，选择 TextChanged，这是将要处理的事件。选择了该事件后，Visual Basic 2008 自动在代码窗口添加了处理该事件的代码体，如图 8-12 所示。可以看到，Visaul

Basic 添加了两行代码，分别是 Private Sub txtInput_TextChanged…和 End Sub，并且光标自动停在了需要添加代码的地方。

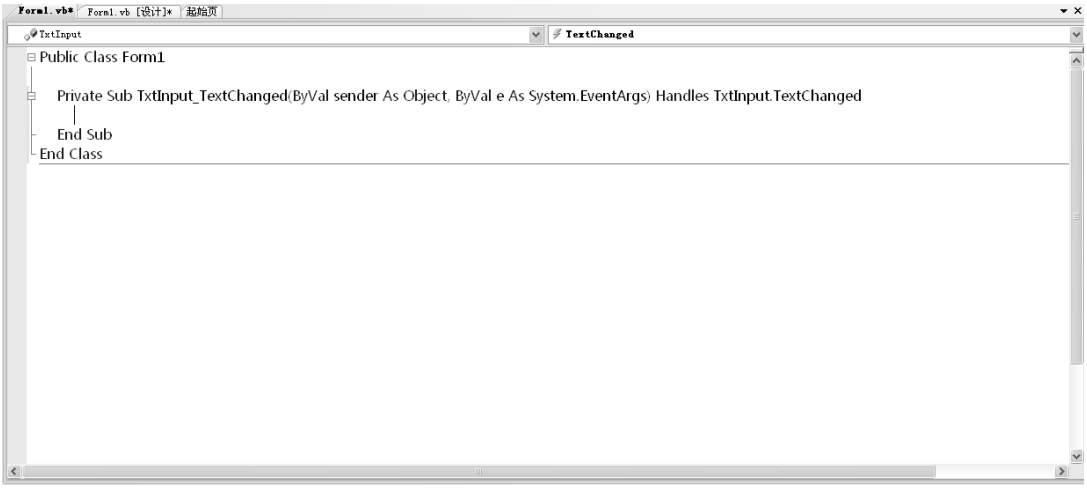


图 8-12　Visual Basic 添加的事件处理代码

程序：

```
    Private Sub TxtInput_TextChanged(ByVal sender As Object, ByVal e As
System.EventArgs) Handles TxtInput.TextChanged
        TxtCopy.Text = TxtInput.Text
    End Sub
```

运行结果：

由于所举的例子较为简单，所以整个程序只需要添加这一行代码就够了。现在运行程序，可以看到，在文本框 Input 的中的任何改变，文本框 Copy 始终和它保持一致，如图 8-13 所示。

图 8-13　复制字符程序的运行结果

上面的程序是如何工作的呢？实际情况是这样的：每当在输入字符的文本框中输入字符或者删除字符时，文本框的内容被改变，此时，系统（Windows）发送 TextChange 消息到应用程序。由于存在有该消息的处理程序，应用程序将调用该处理程序，于是语句

```
    txtCopy.Text = txtInput.Text
```

被执行，输入字符文本框中的内容被复制到字符文本框中。

事实上，程序 HelloWorld 中，为按钮编写代码的步骤与此处是完全一致的。同样也可以通过在设计窗口上单击右键，选择查看代码从而打开代码窗口。在代码窗口的左边的下拉窗口中选择按钮控件的名称 Name，在右边的下拉窗口选择事件 Click。随后在 Visual Basic

生成的代码框架中添加自己的程序。Visual Basic 为每一个控件的最常用事件设计了一个“快捷方式”，即只要在设计窗口双击该控件，Visual Basic 会自动打开代码窗口，同时为这个控件添加这个最常用事件的程序框架。每个控件都有一个最常用事件，如按钮控件是单击事件 Click。前面的编程正是利用了这一特性来快速完成的。每个控件的最常用事件是 Visual Basic 事先设计好的，不能再被改变。如果要编写的事件处理程序不是该控件的最常用事件，那么只好通过 StringCopy 这个程序所用的步骤来添加事件处理代码。

上面的两个例子程序大致讲述了 Visual Basic 编程的一般步骤，总结起来，编写一个 Visual Basic 程序一般有以下几步：

- **运行 Visual Studio**。这无疑是要做的第一步，如果你的机器还没有配置好，请首先安装 Visual Studio。
- **创建新项目**。为项目取一个名字，有必要的话重新选一个保存项目的位置，然后创建这个项目。
- **为窗体添加控件**。将需要的控件从工具箱放到窗体中，排列好它们的位置，完成程序的静态设计。
- **为项目中包含的对象设置属性**。为在上一步添加的控件（对象）设置属性，使控件（对象）符合要求。
- **编写使程序运转起来的代码**。这是最关键的一步，实际的编码在这一步完成，一个发挥想象力和创造力的好地方。
- **程序的生成与运行**。使程序按照设计运行，无疑是最终的目标。

8.1.2 常用控件

程序的界面，就是指程序启动后，用户在屏幕上看到的程序的“样子”，或者说程序的“外观”。Visual Basic 2008 称为可视化的编程环境，主要是指用户在进行界面的设计时，无须编写任何代码，而是通过鼠标的拖拽和对控件属性的设置来完成界面的设计。Visaul Basic 会自动生成相应的程序代码，使得程序启动后所显示的界面和设计时的一样。

控件是可以和用户或程序实现互动的一个对象。绝大多数程序都是可以互动的——它们需要从用户那里获取信息，并向用户反馈信息。基于 Windows 的 Visual Basic 程序通过程序窗体的控件实现与用户的交互。正如例8-2 通过文本框控件得到用户输入的整数，同时利用另一个文本框控件将结果显示给用户。

控件的属性控制着对象的外观和行为。通过对同样的控件设置不同的属性，可以使它们表现出不同的外观和行为。许多属性是每一个控件都有的，还有一些是大部分控件都有的。这些属性对每一个控件来讲，用法是相同的。

将控件添加到窗体后，通常要设置控件的一个或多个属性。对于例8-2 所添加的文本框控件，要设置它的 Name 和 Text 属性。

Name 属性非常重要，在程序代码中，它用来指明控件。由于程序一般有多个同类控件，所以，可以用控件的 Name 属性来唯一标识某一特定的控件。每个控件都必须有名称，其名称用控件的 Name 属性值来表示。除此之外，特定窗体上的每个控件都必须有一个唯一的名称。

Visual Basic 为窗体上放置的每一个控件都分配了一个默认名称，如 TextBox1、

TextBox2 等。更改默认控件的名字使其更加具有现实意义，是一个良好的编程习惯。

在图8-3 所示的属性窗口可以在设置控件的属性。被选中的控件名字出现在属性窗口上部的下拉框中。该控件的属性被分类后列出。左边一栏是属性的名字，右边是属性的值。单击属性的名字，可以在下方看到对该属性的简单提示。要改变属性的值，只需单击原有属性的值，作相应的改变即可。

控件在窗体上应当排列整齐。要移动控件，只需选中该控件，简单的拖动即可。如果想更改控件尺寸，首先必须选中它（单击一个控件即可选中），令其可缩放的控制点显示出来。然后，通过拖动控制点的方式，更改控件的尺寸。控件顶部和底部边缘的控制点用于更改控件的高度。左边和右边的控制点则用于更改控件的宽度。控件四角的控制点同时更改其高度和宽度。

默认状态下，有的控件（如文本框控件）只有两个控制点是可用的（一共有 8 个）。其余的呈灰色不可用。这是因为文本框控件的 AutoSize 属性在默认状态下被设置为 True。AutoSize 属性会根据控件即将显示的文本字体大小自动调整文本框的高度。因此，调整高度大小的控制点不可用。如果需要控制文本框控件的高度，可以将 AutoSize 属性设置为 False，此时 8 个控制点就全部可用了。

另外，还可以在属性窗口中，分别修改 Size 和 Location 属性，从而修改对象的大小和位置。Size 属性有两个值组成分别表示控件的高度和宽度。Location 的两个值则分别表示的是控件相对于容器的 x、y 坐标。

如果控件不是对得很齐，还可以这样做：将要对齐的控件选中，为了做到这一点，可以先选中一个，再按住 Ctrl 键选中其余的。例如，要对齐 lblInput 标签控件和位于它旁边的 TxtInput 文本框控件。首先用上述的方法将这两个控件选中，然后在菜单中选择格式→对齐→中间对齐，如图 8-14 所示。

图 8-14　对齐控件的菜单

Visual Basic 提供了很多控件，在此无法一一列举。请读者参阅其他专门的编程书籍，下面我们简单介绍几个较为常用的控件。

（1）Label（标签）控件：为控件和窗体的其他组成部分提供标识。使用 Label，可以给用户提供出窗体功能的有关信息。从广义上说，窗体中的每一条文字都是一个 Label 控件。

（2）TextBox（文本框）控件：一个应用程序中会多次用到该控件。TextBox 控件的应用范围非常广，例如，可用来显示一个由多行文本组成的版本信息。实际上 TextBox 能容纳的文本数量是没有限制的，当文本数量超出文本框的尺寸时，文本框还会添加自己的滚动条。TextBox 和 Label 控件之间的差别在于 TextBox 控件中的文本可以被编辑，而 Label 控件中的文本不能被编辑。

（3）Button（按钮）控件：用户可以单击按钮控件触发程序动作。

（4）RadioButton（单选按钮）控件：用来让用户在一组选项中选定一项且只能选定一项。若窗体内仅有一组选项按钮控件时，可将它们直接放置在这个窗体内即可。但当有两组或多组选项时，RadioButton 应该被放置到一个 GroupBox 控件（下面将要介绍）内。

在窗体或框架内创建一个 RadioButton 时，单击 RadioButton 对象，鼠标箭头变为十字

形状，将箭头移至窗体上或框架内的合适位置，按住左键，拖动鼠标。到适当大小时，释放左键，如图 8-15 所示。

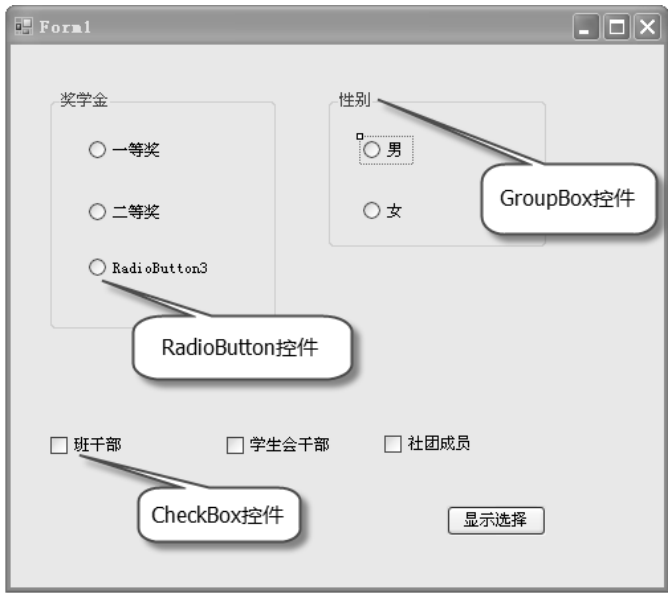


图 8-15　RadioButton、CheckBox 和 GroupBox 控件

RadioButton 有许多属性，其中最常用的属性如下：

- Text 属性。设定 RadioButton 旁边的文本内容。
- CheckAlign 属性。CheckAlign 属性设定控件按钮与文本的位置关系。
- Checked 属性。由 Checked 属性设定 RadioButton 的状态。True：RadioButton 被选定；False：RadioButton 未被选定，默认设置。

（5）CheckBox（复选框）控件：是让用户在一组选项中选定一项或选定多项。若窗体内仅有一组 CheckBox 控件时，可将它们放置在这个简单的窗体内，但当有两组或多组 CheckBox 时，CheckBox 通常也被放置到一个 GroupBox 控件内。

CheckBox 的属性中也有 Text 属性、CheckAlign 属性和 Name 属性等，这些属性的用法几乎每个控件都大同小异。CheckBox 最重要的属性是 Checked 属性。通过 Checked 属性可以检查或设定 CheckBox 是否被选中：

- Checked = True。被选中。
- Checked = False。未被选中。

CheckBox 中还有一个 CheckState 属性用来指示 CheckBox 目前的状态：

- CheckState = Checked。被选中状态。
- CheckState = UnChecked。未被选中状态。
- CheckState = Indeterminate。不可用状态（当 ThreeState 属性设置为 true 时有效）。

（6）GroupBox 控件，RadioButton 控件：在一组中只能选定一项，怎样对 RadioButton 控件分组呢？其中一个方法就是利用 GroupBox 控件（GroupBox 控件在工具箱的“容器”组中）。可以先将一个 GroupBox 控件放置在窗体上，然后将 RadioButton 控件放在 GroupBox 控件即可。在一个 GroupBox 控件中的 RadioButton 控件自动成为一组。另外，

还可以设定 GroupBox 的 Text 属性，对分组做一个说明，如图 8-15 所示。

CheckBox 也可以放在一个 GroupBox 控件中，但 CheckBox 并没有类似 RadioButton 分组的类似概念。将 CheckBox 放入一个 GroupBox 的作用主要是说明和装饰作用。

（7）Timer 组件：是按标准时间间隔引发计时器事件的组件。该组件是为 Windows 窗体环境设计的，Timer 组件的主要属性如下：

- Enabled 属性。设置定时器允许或禁止产生计时信号。
- Interval 属性。Timer 就好像家里的闹钟，设定的时间一到就会动作，只不过定时器的功能比闹钟广一点。可以用 Interval 属性设定 Timer 的动作间隔，每当设定的时间间隔一到，系统会自动产生一个计时信号，以激活 Timer 的 Tick 事件。Interval 的值需介于 1～65535，每单位为 1 ms，表示每 1ms 会激活 Timer 事件一次。最大的时间间隔约为 1.5min。在程序运行期间，Timer 组件永远是不可见的（Invisible）。如果计时间隔（Interval）比 Timer 程序的执行周期还短，会发生上一个 Timer 程序还在执行中，但计时间隔已到，下一个 Timer 事件又发生的现象。此时，会将后来这个 Timer 事件忽略掉，直到上一个 Timer 程序执行完毕，才会再接受另外一个 Timer 事件。

Tick 事件，若启用了该组件，则每个时间间隔引发一个 Tick 事件。就如同左键单击发送 Click 事件一样。如果为 Tick 事件编写了处理代码，则这段代码会每隔一定的时间就被执行一次。

8.1.3　编程实例

例 8-3　编写一个倒计时程序，用户输入一个分钟数，单击 go 按钮后，程序开始倒计时，以十分之一秒为单位。

步骤：

（1）新建项目 CountDown。

（2）在窗体上放置 2 个 Label 控件、2 个 TextBox 控件、1 个 Button 控件和 1 个 Timer 组件。分别设置属性如表 8-5～表 8-10 所示，程序界面如图 8-16 所示，Timer 组件本身没有界面，它被专门显示到了下方。当单击 Go 按钮时 Timer 组件启动，为此只需将 Timer 的 Enbled 属性设为 True。

表 8-5　Label1 的属性

属性	值
Name	LblInput
Text	输入倒计时的分钟数
TextAlign	BottomLeft

表 8-6　Label2 的属性

属性	值
Name	LblCountDown
Text	倒计时显示
TextAlign	BottomLeft

表 8-7　TextBox1 的属性

属性	值
Name	TxtboxMin
Text	

表 8-8　TextBox1 的属性

属性	值
Name	TxtboxShow
Text	
ReadOnly	True

表 8-9　Timer 的属性

属性	值
Name	TimCount
Interval	100

表 8-10　Button 的属性

属性	值
Name	BtnCount
Text	Go

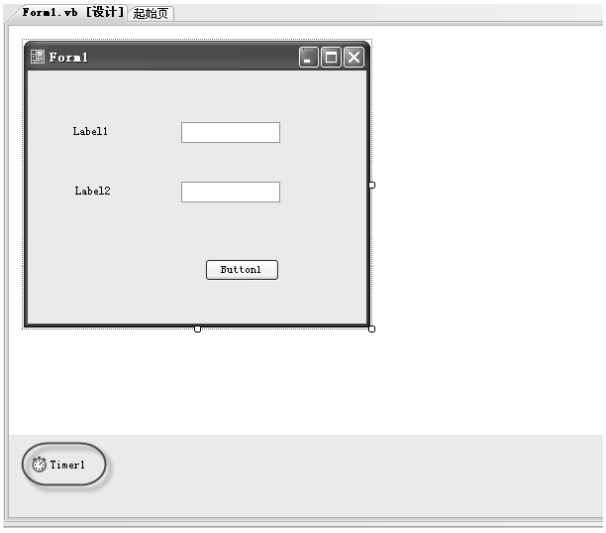


图 8-16　Timer 控件

（3）为按钮的 Click 事件编写代码如下：

```
Dim intMinutes As Integer
Dim lngTenSecond As Long

Private Sub BtnCount_Click(ByVal sender As System.Object, _
ByVal e As System.EventArgs) Handles BtnCount.Click

       intMinutes = Convert.ToInt32(TxtboxMin.Text)

       If intMinutes > 0 Then
          TimCount.Enabled = True
          lngTenSecond = intMinutes * 600
       End If
End Sub
```

我们将变量 IntMinutes 和 lngTenSecond 的定义写到了 Private sub … End Sub 块的

外面，把它们放到了更高一级的块 Public Class … End Class 中。因此，这两个变量的作用范围被扩大了，这样做的原因是在 Timer 的 Tick 处理程序中也要用到这两个变量，同时还要保持它们的值。具体地讲，IntMinutes 保存了用户输入的分钟数；lngTenSecond 保存了倒计时还剩下的时间数，以十分之一秒为单位。

（4）为 Timer 的 Tick 事件编写代码如下：

```
    Private Sub TimCount_Tick(ByVal sender As Object, ByVal e As System.EventArgs)
Handles TimCount.Tick
        Dim intMin As Integer
        Dim intSec As Integer
        Dim intSecTen As Integer

        lngTenSecond -= 1

        If lngTenSecond >= 0 Then
            intMin = lngTenSecond \ 600
            intSec = (lngTenSecond - (intMin * 600)) \ 10
            intSecTen = lngTenSecond Mod 10

            TxtboxShow.Text = intMin.ToString() + ":" + intSec.ToString() _
            + ":" + intSecTen.ToString()
        Else
            TimCount.Stop()
        End If

    End Sub
```

根据设定，Tick 事件在 Timer 被启动后每十分之一秒发生一次。因此，Tick 事件的处理代码会每十分之一秒被执行一次。每次将剩余的时间减 1，然后将剩下的时间按分钟、秒和十分之一秒显示出来。当时间剩余为 0 时，关闭 Timer。为了使每次执行这段代码时，lngTenSecond 的值不被破坏，也需要把 lngTenSecond 定义在此函数之外。程序运行结果如图 8-17 所示。

图 8-17　倒计时的运行结果

8.2　网络编程

通过 Internet 访问的大部分信息都是存储在称为“服务器”的计算机上。服务器可以是任意一种类型的计算机，使它成为服务器的原因是它所起的作用是存储着可供客户机使用的数据。

“客户机”是一台计算机，更确切地说，是一个特殊的计算机程序，它知道如何与某种类型的服务器通信以便使用服务器上存储的信息（或者把信息存入服务器）。例如，当在 Web 上冲浪时，会使用一种称为 Web 浏览器的客户机程序（如 IE）与存储 Web 页的计算机（Web 服务器）通信。Web 浏览器（Web Browser）是一个赋予计算机与 We b 服务器通信并显示服务器上存储的信息的程序。

一般而言，每一种类型的 Internet 活动都涉及不同的客户机和服务器类型。要使用 Web，就需要使用 Web 客户机程序与 Web 服务器通信；要使用电子邮件，就需要使用电子邮件程序与邮件服务器通信。

这种客户机和服务器的关系表明：Internet 实际上只是一种通信媒介，计算机之间的通信是通过一种虚拟线路实现的。决定实现各种活动的是各种类型的客户机和服务器，而不是 Internet 本身。因为会出现新的客户机和服务器类型，所以，新的活动类型可能会随时被添加到 Internet。

8.2.1 客户端编程

例 8-4 在这一节里，利用 IE 的组件，创建一个简单的 Web 浏览器的客户端。具体地讲，将使用 WebBrower 控件来完成程序。可以使用该程序来从 Internet 服务器获取 Web 页，并将 Web 页显示到屏幕上。

步骤：

（1）新建项目 MyIE。需要使用的 WebBrower 控件并不在工具箱的默认设置里。因此，将该控件加入到工具箱中。首先切换到工具箱的组件栏，在该栏的空白处右击，从弹出的菜单中选择“选择项”。将看到如图 8-18 的窗口，在 COM 组件页中选择 Microsoft Web Browser，然后单击“确定”按钮。

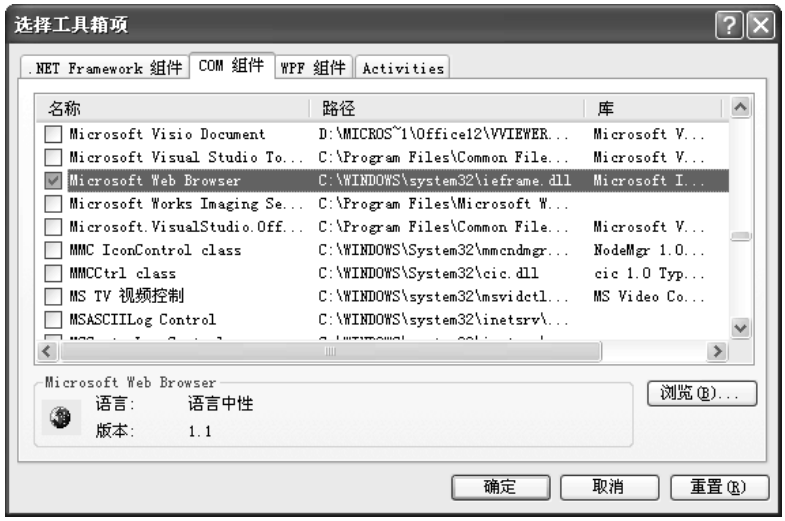


图 8-18 添加 WebBrower 组件

（2）添加后可以在工具栏的组件里看到 Visual Basic 添加了一个组件 Exploer，带有一个地球的图标。将这个新添加的组件拖放到 Form 窗口中，这是一个可视的组件，在窗体上显示为一个白色的矩形区域。一个 Web 页将显示在该控件中。

（3）接下来再放置 3 个按钮，分别用于后退、前进和开始。最后放置一个文本框用于输入 Internet 地址。合理的命名这些控件的 Name 属性。为了使 Web 页在窗口缩放的时候自动随之缩放，可以使用控件的 Anchor 属性将控件和窗体之间的相对位置固定。

（4）具体的，3 个 Button 控件的 Anchor 属性为 Top，Right；TextBox 控件的 Anchor 属性为 Left，Top，Right；Microsoft Web Broswer 的 Anchor 属性为 Left，Top，Right，Down；

设计好的界面如图 8-19 所示。

图 8-19　简易浏览器的界面

对 Explorer 组件来讲，该组件和我们前面使用过的其他控件一样，有自己的方法、事件和属性。其中一些常用的方法如下：

- Navigate2。该方法接收一个 URL 地址参数，将该 URL 指定的 Web 页在组件中显示出来。
- GoBack。向后回退一个网页。
- GoForward。向前前进一个网页。
- GoHome。显示主页。
- ReFresh。刷新当前页。
- Stop。停止当前页的显示和下载。

常用的事件如下：

- BeforeNavigate2。将要开始下载并显示一个 Web 页。
- NavigateComplete2。Web 页下载显示完毕。
- FileDownload。将有文件下载操作发生。

常用的属性如下。

- LocationURL。指示当前 Web 网页的 URL 地址。

（5）了解了这些属性和方法后，继续例 8-4 的编写。首先转到按钮编写代码，当单击该按钮时，程序从文本框得到 URL。然后调用 Navigate2 方法来打开一个网页。

代码：

```
    Private Sub BtnGo_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BtnGo.Click
            WebBrowser.Navigate2(TxtAddress.Text)
```

```
    End Sub
```

（6）前进和后退按钮编写代码如下：

```
    Private Sub Btnforward_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles Btnforward.Click
           Try
              WebBrowser.GoForward()
           Catch ex As Exception
           End Try
    End Sub

    Private Sub BtnBack_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BtnBack.Click
           Try
              WebBrowser.GoBack()
           Catch ex As Exception
           End Try
    End Sub
```

运行结果：

有了以上代码后，我们的程序已经可以工作了。注意到在前进或后退的代码使用了异常捕获，虽然捕获后什么也未处理，但却防止了用户一直后退或前进到一个不存在的网页上而使程序出错。在文本框内输入网址，网页会显示在窗口中。

同时注意到程序还有一些不完善的地方，如在窗口单击其他链接后，网页改变了，文本框中的地址却不会变化；在文本框中输入地址后按回车键却没有反应。下面我们来添加代码改变这两点。首先对文本框的 KeyDown 事件编写代码，每次在文本框中有键按下时将产生该事件。当检测到按键是回车（Enter）时，装载新的网页。

当网页显示完毕时，Explorer 控件会产生 NavgateComplete2 事件，此时更新 TextBox 控件中的显示：

```
    Private Sub TxtAddress_KeyDown(ByVal sender As Object, ByVal e As
System.Windows.Forms.KeyEventArgs) Handles TxtAddress.KeyDown
           If (e.KeyCode = Keys.Enter) Then
              WebBrowser.Navigate2(TxtAddress.Text)
           End If
    End Sub

    Private Sub WebBrowser_NavigateComplete2(ByVal sender As Object, ByVal e As
AxSHDocVw.DWebBrowserEvents2_NavigateComplete2Event) Handles
WebBrowser.NavigateComplete2
           TxtAddress.Text = WebBrowser.LocationURL
    End Sub
```

将以上两条语句加入后，基本上就可以作为一个简单的浏览器来用了。但是更主要的是，我们有了一种选择，在自己的程序里显示特定的网页。

8.2.2 ASP 编程概述

ASP（Active Server Page）是微软公司研发的一种交互式网页编程技术，从 1996 年发

布的 ASP 1.0 开始，ASP 开始从实验室走向实际应用中，但是并没有为人们所追宠。1998 年微软发布了 ASP 2.0。2000 年，微软公司发布了它的革命性的服务器系统 Windows 2000，该系统上集成了 IIS 5.0，并捆绑了 ASP 3.0。由于 ASP 提供了一系列的 Web 应用程序组件可以用来执行高级功能（如 ADO 对象来实现对数据库的操作），再加上本家系统的稳定支持，因此，Windows 2000+ASP 3.0 便成了当时最流行的 WWW 服务器模式，ASP 也因此在全球风靡起来。ASP 的编程语言为 VBScript 和 JavaScript，运行机制是解释型的。ASP 页面文件的后缀名为.asp，当客户机提交访问时，Web 服务器就找到该页面，并交给解释引擎对 ASP 页面执行一次解释，并把结果发送给客户机。在当时，这种技术是具有先进性的。但是随着 WWW 服务的广泛应用，越来越多的 Web 应用程序应用到 WWW 服务上，解释型的 ASP 技术在处理大型 Web 程序和频繁访问时，给服务器带来瞬间几何级系统开销，因此，ASP 的改进就显得很必要了。

2001 年，微软公司推出了 ASP.NET。从命名上看，可以说 ASP.NET 是 ASP 3.0 的升级，实际上 ASP.NET 是一种全新的交互式网页编程技术，是网站和 XML Web 服务的产物，也是微软公司新的应用开发平台——.NET 框架中的核心要素。如果说微软公司的.NET 计划是编程技术的一种革命，那么，ASP.NET 则无疑是 ASP 的一种革命，ASP.NET 技术把面向对象的编程技术引入到 Web 编程中，这使得在编制 Web 应用程序的时候，就更像与在编制 Windows 应用程序一样的简便快捷。

ASP.NET Web Forms 页面是以 aspx 为扩展名的文本文件。它们可以通过 IIS 虚拟根目录树来进行配置。当浏览器客户端请求.aspx 资源时，ASP.NET 运行时刻库分析和编译目标文件，形成.NET 框架类。这个类能够用来动态的处理即将开始的请求（注意：.aspx 文件只有在第一次被访问的时候编译；编译后的结果在以后的请求中被重复利用）。

可以将 ASP.NE 页面简单的看成一般的 HTML 页面，页面上包含标记有特殊功能的段。ASP.NET 模块分析 ASPX 文件的内容，并将文件内容分解成单独的命令以建立代码的整体结构。完成次工作后，ASP.NET 模块将各命令放置到预定义的类中。然后这个类被用来定义一个特殊的 ASP.NET Page 对象。该对象要完成的任务之一就是生成 HTML 流，这些 HTML 流将被返回到客户。

8.2.3 ASP.NET 编程简介

所有 ASP.NET 页面以.aspx 为扩展名，这一点非常重要，因为所有 ASP.NET 页均由添加给文件名的.aspx 后缀来确认。只有以.aspx 为后缀的页面才能够送到 ASP.NET 进行处理，甚至是以.aspx 为后缀的纯 HTML 页面也会被送到 ASP.NET 处理。.aspx 后缀名是唯一确认一个页面是 ASP.NET 页的标志。

ASP.NET 的代码可以有两种方式来编写：一种是将代码直接放入到 ASP.NET 的页面中，另一种是将页面和代码分开。

先来看前一种编程方式，在这种编程方式下，甚至都不需要 Visual Studio，一个记事本有时便足够了。将 ASP.NET 代码插入到自己的 Web 页源代码中，需要对其进行标注，以使服务器能将它确认为服务器端代码，使它与 HTML 代码有所区别。在自己的页面中将 ASP.NET 代码与 HTML 代码区别的最好方法是使用<Script>标识符，并将 Runat 属性设成 Server。该设置表明处理代码的目标主机是 Web 服务器。当采用<Script>标识符时，脚本默

认在客户端执行，所以 Runat 属性必须设置。

此外，还可以使用<%和%>标记，表明服务器端代码的开始和结束。但是，如果要使 ASP.NET 正确工作，对函数声明使用<Script>标记，而对页面处理过程中需要处理的语句使用<%和%>标记。一个简单的示例如下：

```
<Script Language="VB" Runat="Server">
    Sub MyTest( )
        Response.Write(“Hello”)
    End Sub
</Script>

<%
    Call MyTest
%>
```

ASP.NET 本身不是语言，是创建动态页面的技术。它允许人们用功能完善的编程语言在自己的页上定义代码段。在 ASP.NET 中编写代码的默认语言是 VisualBasic。用 VisualBasic 定义页面，只要将 Page 指令包括在文件的顶部即可，示例如下：

```
<%@Page Language="VB"%>
```

由于已经将 VisualBasic 确定为默认编程语言，因此，<Script>代码中的 language 属性是可选择的。如果希望使用不同的语言定义页，如 C#，可以采用如下的方式：

```
<%@ Page Language="C#%>

<Script Language="C#" Runat="Server">
    C#的代码…
</Script>
```

此外，Page 指令还可以指定其他的选项，以通知 .NET 如何对页面处理。例如，如果希望处理后生成的 HTML 使用简体中文字符集，则可以加入 CodePage 属性，其中代码 936 表示简体中文（GB2132）：

```
<%@Page Language = "VB" CodePage = "936" %>
```

现在建立一个新文件 Welcome.aspx 并加入一些 ASP.NET 脚本。可以使用任何一个文本编辑器如记事本（Notepad）来新建 Welcome.aspx 文件。

代码：

```
<HTML>
<Body>
<FONT SIZE = "+2" COLOR = "BLUE"> Asp .NET 示例 </FONT> <BR>
<%
Dim nCounter AS Integer

For nCounter = 1 to 10
    Response.Write("<B> 欢迎进入 Asp .NET 编程！ </B> <BR>")
Next

Response.Write("创建时间：" & DateTime.Now)

%>
```

```
</Body>
</HTML>
```

将该页在 IIS 中“发布”也就是复制该页到 IIS 的一个虚拟目录中，假设 IIS 安装在本机，虚拟目录的名字为 ASPTest。打开 IE 浏览器，在浏览器的地址栏中键入地址 http://localhost/ASPTest/Welcome.aspx，结果如图 8-20 所示。如果单击浏览器的刷新按钮，应该可以看到在页面最底行显示的时间将发生变化。这是因为服务器动态的生成了现实时间的代码。再一次的，我们看到 ASP.NET 页面具有以下一些特征：

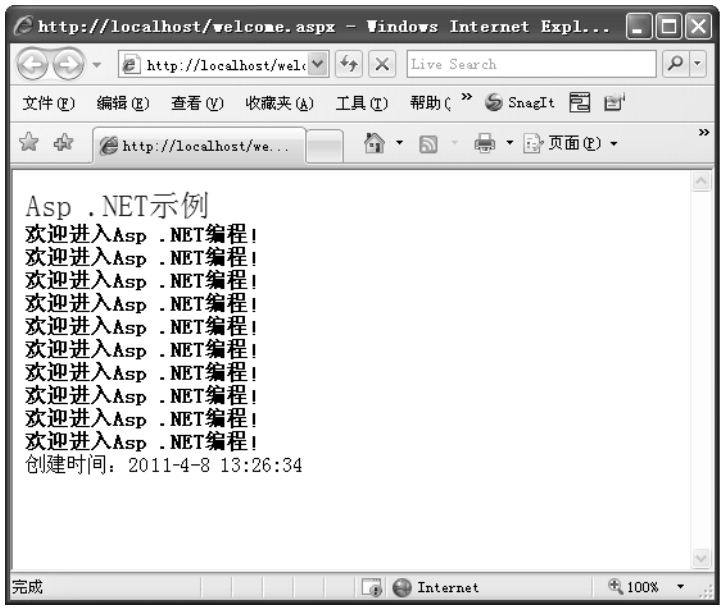


图 8-20　简单的 Welcome.aspx 页面

- ASPX 文件中的脚本标记<%和%>表示位于这之间的内容是由服务器执行的 Visual Basic 代码，以及由<Script Language=“VB” Runat=“Server”>标记的内容将由服务器处理。文件中的其他任何内容，如标准的 HTML 文本，将由服务器直接返回给客户浏览器。
- 在 ASP.NET 中包含几个内建的对象，可以用来操纵 Web 页面的内容。在上面的例子中，Response.Write 方法用来将 HTML 字符串返回给客户浏览器。
- 客户浏览器无法察看服务器端的代码。如果在浏览器中选择查看源代码，只能看到由服务器生成的 HTML 内容，看不到代码本身。

尽管示例较小，但它使得某些动作成为可能。由于 ASP.NET 允许根据 Visual Basic 代码动态的创建页面，因而可以执行一些更为复杂的动作，如连接数据库和返回包含字段值的 HTML 内容。

例 8-5　在 Web 窗体中显示简单的文本和 Calendar（日历）控件。

步骤：

（1）启动 Visual Studio 2008 后，从文件菜单选择新建→网站。选择 ASP.NET 网站，位置使用默认的位置即文件系统选项。选择使用 Visual Basic 语言。命名站点的名字为 Calendar

后单击“确定”按钮，如图 8-21 所示。

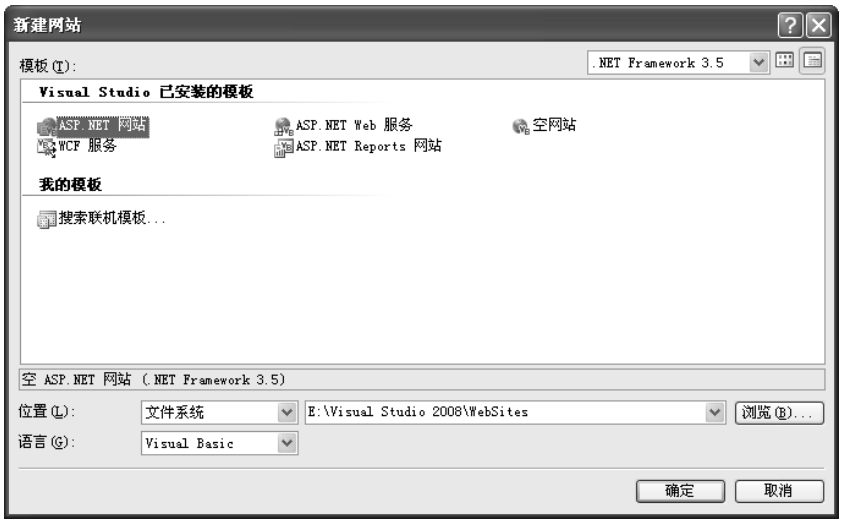


图 8-21　新建一个网站

（2）Visual Studio 将打开网站的默认网页 Default.aspx，并以源视图的方式显示它。在解决方案中包含了一个 default.aspx 的文件和一个 App_Data 的文件夹。

（3）单击代码窗口下的“设计”标签，将视图切换到设计视图。将一个 Label 控件和一个 Calendar 控件从工具箱中拖动到窗体上(可以用回车输入空行，使 2 个控件有些间距)。

（4）在 Label 控件上右击，选择属性，在属性窗口将 Label 控件的 Text 属性改为“以下是一个日历的示例”，如图 8-22 所示。

（5）运行程序。如果在运行时提示要修改 Web.config，则进行修改。

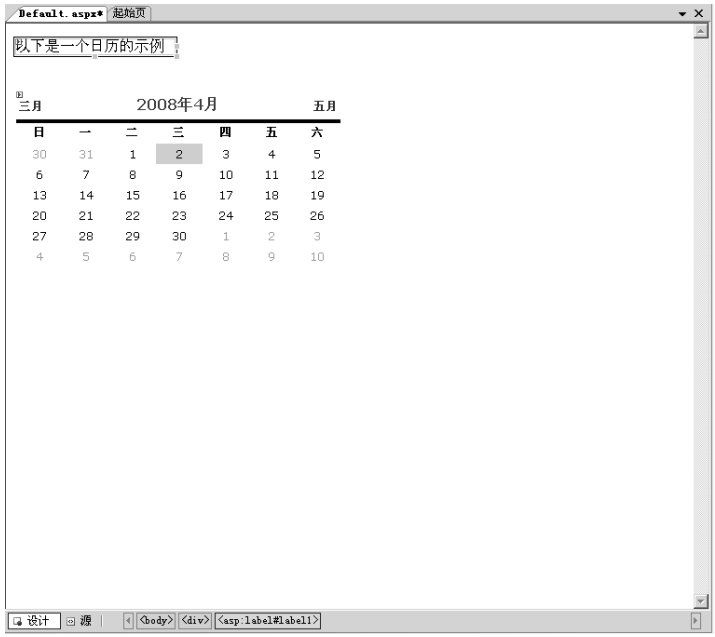


图 8-22　设计好的 Web 窗体

如果没有错误，将在 Web 浏览器中看到运行的结果如图 8-23 所示。

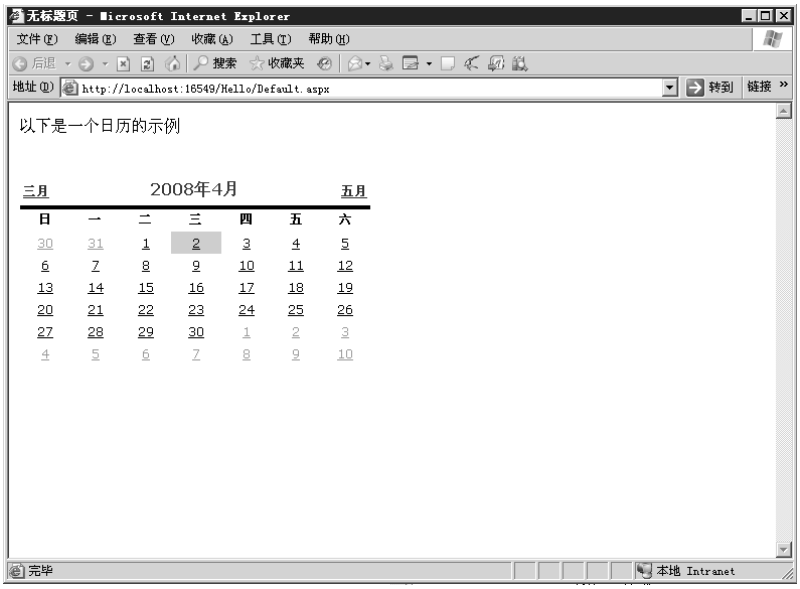


图 8-23　程序的运行结果

8.3　数据库编程初步

数据库是指以文件形式按特定的组织将数据保存在存储介质上，在数据库中，不仅包含数据本身，也包含数据之间的联系，它有如下特点：

- 数据通过一定的数据模型进行组织，从而保证有最小的冗余度，常见的数据模型有层次模型、网状模型和关系模型。
- 数据对各个应用程序实现共享。
- 对数据的各种操作（如定义、操纵等）都由数据库管理系统统一进行。

简单地说，数据库是数据的集合，它是由一个或多个表组成。每个表中都有存储了对一类对象的数据描述。数据库本身不是独立存在的，在实际应用中，用户面对的是包括数据库在内的数据库系统，即具有管理和控制数据库的功能的计算机系统。

8.3.1　数据库系统

如同使用高级语言编写的应用程序要通过解释程序或编译程序来翻译执行一样，数据库的创建和查询也需要使用特定的数据库语言，并需要一种称为“数据库管理系统”的软件的支持才能进行，因此，DBMS 是为数据库的创建、作用和维护而配置的系统软件。它是数据库系统的核心。

数据库管理系统是在操作系统基础上运行的一种支撑软件，它除了像一般语言处理软件，如 Pascal 语言、C 语言编译器等，要对数据库命令和应用程序进行解释执行之外，还需帮助操作系统对数据库实行统一的管理和控制，对多用户数据库提供数据安全性保护等。

数据库管理系统是实现对数据库进行管理的软件，它以统一的方式管理和维护数据库，

并提供数据库接口软件供用户访问数据库。

数据库应用程序是通过 DBMS 访问数据库中的数据、并向用户提供数据服务的程序，即它们是允许用户插入、删除、修改并报告数据库中数据的程序。应用程序是系统开发人员利用数据库系统资源开发的、应用于某一个实际问题的应用软件。程序设计人员使用通过或专用的程序设计语言，如 C 语言、DBMS 自含的语言，以及各种面向用户的数据库应用程序开发工具等，按照用户的要求编写的。

8.3.2 使用 Visual Studio 操作数据库

在 Visual Basic 中，可以直接从开发环境中访问可视化数据库工具，来创建、管理数据库对象。这样，就不必转到外部程序进行管理，如 Access 或 SQL Server Enterprise Manager 等。Visual Studio 包含有数据库项目，允许在解决方案资源管理器中管理数据库查询和 SQL 脚本。在 Visual Studio 中新建项目，在新建项目对话框中选择其他项目中的数据库项目，如图 8-24 所示。

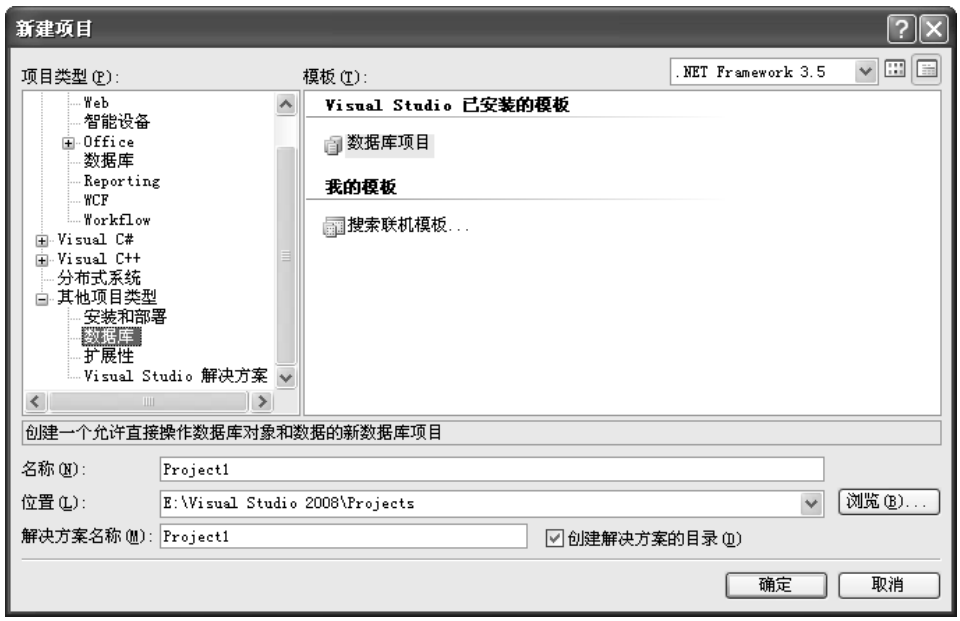


图 8-24 新建数据库项目

为了简化数据库的连接工作，Visual Studio 采用数据库引用的方式。数据库引用是对某个特定数据库的连接信息，该信息能被存储在一个 Visual Studio 项目中。如果没有任何引用，则会自动出现选择或更改数据源的对话框如图 8-25 所示。随后建立连接的对话框出现，建立连接的对话框随数据源选择的不同而不同。

如果已经建立过连接，则可以使用前面的连接（会先出现一个对话框选择已有的连接），或者建立一个新的。新建连接的对话框如图 8-26 所示。

提供程序选项卡用来说明是什么类型的数据库，SQL Server 是默认设置。在连接选项卡输入各项后单击测试连接，通过后单击“确定”按钮。数据库项目被创建。

建立连接后便在服务器资源管理器创建了一个数据库的连接，如图 8-27 所示。

图 8-25　选择或更改数据源

图 8-26　新建数据库连接的对话框

图 8-27　在服务器资源管理器中查看数据库

在建立了到数据库的连接之后，使用服务器资源管理器可以执行以下操作：

- 可以右击表文件夹，从快捷菜单中选择新建表命令，打开表设计器，如图 8-28 所示，创建的新表保存在数据库中。
- 可以通过单击一个数据表，然后按 Del 键，或右击该表，从快捷菜单中选择删除命令来删除一个表。
- 可以右击一个表，从快捷菜单中选择设计表命令，出现表设计器，在此对表进行修改，然后保存对表所做的修改。
- 双击一个表或视图，或者右击，从快捷菜单中选择从表（视图）中检索数据命令，

来查看一个表或视图的内容，如图 8-29 所示。如果有相应的权限，还可以编辑网格中的数据并把修改保存在数据库中。

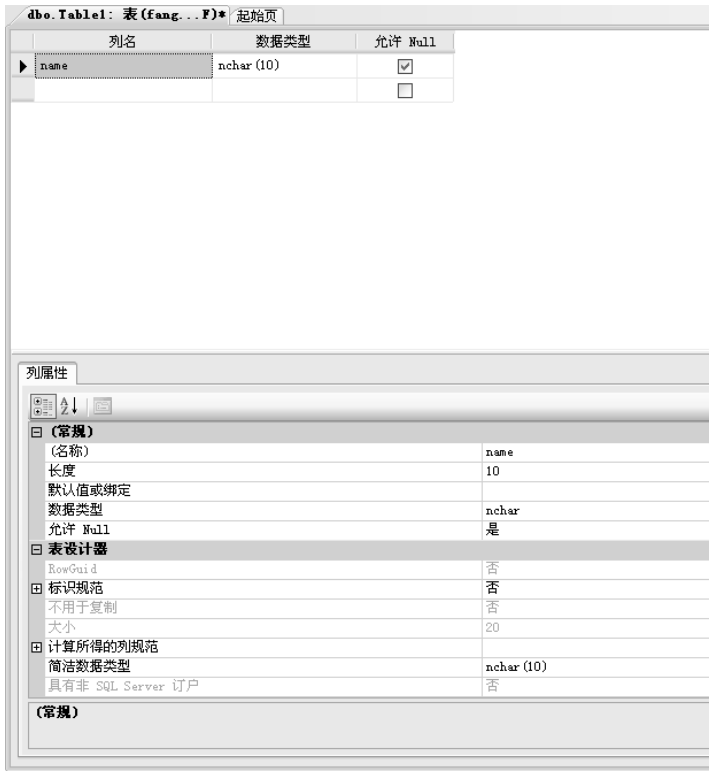


图 8-28　Visual Studio 的表设计器

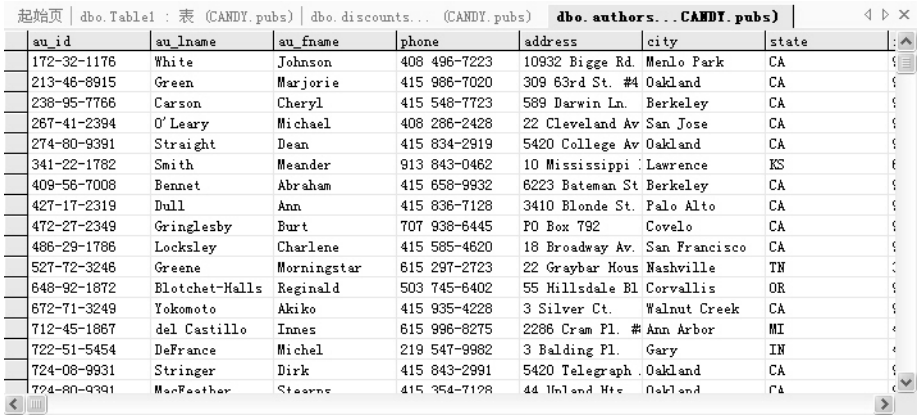


图 8-29　可以直接编辑表中的数据

8.3.3　在 Visual Basic 中访问数据库

当需要从 Visual Basic 程序中访问数据库时，Visaul Basic 程序是数据库的客户端。在访问数据库之前首先需要连接到数据库。

在这里，以 Microsoft 的 SQL Server 为例，使用的是 pubs 数据库。为了方便管理到数

据库的连接，Visaul Studio 提供了 SqlConnection 对象，表示 SQL Server 数据库的一个打开的连接。该对象在 System.Data.SqlClient 名称空间中，在使用前需要 Imports 该名称空间。

可以通过如下的代码创建一个到 SQL Server 的连接：

```
Dim strInfo As String
strInfo = "data source=CANDY;initial catalog=pubs; user id=sa;password=sacsn; _
          workstation id=CANDY;packet size=4096"
Dim sqlcnMyDB As New SqlConnection(strInfo)
```

上述代码首先声明了一个连接字符串，该字符串的每一项以分号隔开，分别表示了数据库服务器的计算机名、用户名和密码等信息。也可以将一个 SqlConnection 组件从工具箱中拖动到窗体上的方法来创建一个连接。SqlConnection 组件在工具箱的数据窗格中。之后可以选中该组件然后在属性窗口设置 ConnectionSting 属性，也就是填入和 strInfo 相同的内容。

然而，还有更加简单的方法来创建一个连接。打开服务器资源管理器，找到数据连接项，如果在该项下面已经存在一个需要的连接，则直接将该连接由服务器资源管理器拖到窗体上即可。如果没有，可以右击选择新建连接，将出项在上一章介绍的新建连接对话框。建立好连接后将它拖到窗体上即可。

一旦将连接拖动到窗体上，Visual Basic 将在后台生成代码连接到数据库。此时一个 SqlConnetion 组件将显示在窗体的下方。可以在属性窗口对该连接做进一步的设置，如更改 Name 属性。如果数据库服务器设有密码，也许需要手工将 password =项添加到 ConnectionString 属性中。

在对数据库实际操作之前，首先需要打开连接，使用完后应立即关闭，这是由 Open 和 Close 方法来完成的：

```
sqlcnMyDB.Open( )

'对数据库操作
sqlcnMyDB.Close( )
```

实际对数据库的操作可以使用 SqlCommand 对象来完成。可以在代码中声明一个 Command 对象，也可以从工具箱中将一个 SqlCommand 对象拖到窗体上。要使用 SqlCommand 对象对数据库进行操作，首先应该设置它的 CommandText 属性。该属性是一个字符串，实际就是一个完整的 SQL 语句。代码如下：

```
sqlcmMyCommand.CommandText = "SELECT au_lname FROM 学生"
```

设置好 CommandText 属性后需要指出该对象使用哪个连接，代码如下：

```
sqlcmMyCommand.Connection = sqlcnMyDB
```

如果 SqlCommand 对象是由工具箱拖动到窗体上的，也可以在它的属性窗口设置 Connection 属性。接下来将调用语句执行该 SQL 语句。根据 SQL 语句的不同，需要执行不同的调用方法。

- ExecuteNonQuery。执行一个不返回记录的 SQL 语句，如 DELETE 和 UPDATE 语句。
- ExecuteReader。执行一个 SQL 语句并返回一个 SqlDataReader 对象，该对象包含了结果记录。
- ExecuteScalar。在只需要从 SQL SELECT 语句返回一个字段的值时使用。

这样做是为了提高数据库的效率。

例 8-6　对数据库 students 进行查询结果显示的操作。

分析：首先建立一个项目，在项目中若需对数据库进行操作。则要建立该项目与数据库的连接。然后，在项目窗体上添加必要的控件和进行相应设置，将当前项目与数据库建立连接。最后，编写程序。

步骤

（1）建立新的项目，命名为“Query”。

（2）因为数据库 students 是在 Visual Studio 中创建的，所以，打开“服务器资源管理器”窗口，找到 students 后，将连接由服务器资源管理器拖入项目窗口内即可（若是所用数据库为 Visual Studio 以外的程序所建立的，则需要建立连接。连接属性对话框如图 8-30 所示）。

（3）在窗体上放置一个 ListBox 控件和一个 Button 控件。

（4）设置 ListBox 和 Button 的 Name 属性分别为 LstResult 和 BtnQuery。Button 的 Text 属性设置为 Go。界面如图 8-31 所示。

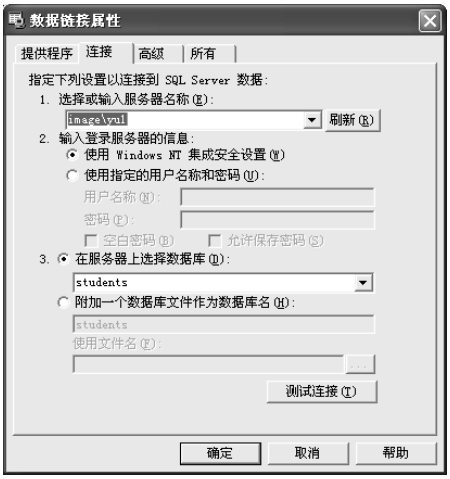


图 8-30　创建与数据库 students 的连接

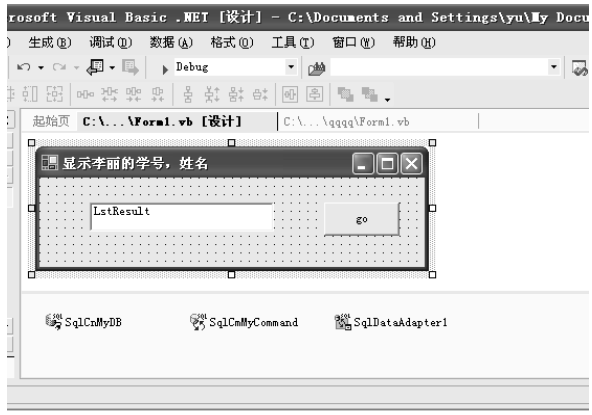


图 8-31　设计好的界面

（5）从工具箱中将 SqlConnection 和 SqlCommand 控件拖动到窗体上，设置对应的 Name 属性分别为 SqlCnMyDB、SqlCmMyCommand 和 Button 控件的属性设置如表 9-11 所示。

表 8-11　控件的属性设置

控件或窗体	属性	值
显示内容的窗体 Form	Text	显示李丽的学号，姓名
显示查询结果的 ListBox 控件	Name	LstResult
SqlConnection	Name	SqlCnMyDB
SqlCommand	Name	SqlCmMyCommand
Button	Name	BtnQuery
Button	Text	Go

（6）双击 BtnQuery 控件，添加按钮代码。

程序：

```
    Imports System.Data.SqlClient
    Private Sub BtnQuery_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles BtnQuery.Click
```

```
      SqlCmMyCommand.Connection = SqlCnMyDB
     SqlCmMyCommand.CommandText = "SELECT 学号,姓名 FROM 学生 WHERE (姓名= '李丽
')"
    '用 SQL 语句查询姓名为李丽的记录

         Dim drStudent As SqlDataReader
         Dim strName As String
           SqlCnMyDB.Open()  '将连接的数据库 students 打开
           drStudent = SqlCmMyCommand.ExecuteReader()

         While drStudent.Read()
          strName = drStudent.GetString(0).Trim + " " + drStudent.GetString(1).
Trim
          LstResult.Items.Add(strName)
         End While

         drStudent.Close()
         SqlCnMyDB.Close()'将连接的数据库 students 关闭

        End Sub
    End Class
```

程序运行结果如图 8-32 所示。

图 8-32　显示学生李丽的学号，姓名的结果

附录 A　常用外设及设备驱动程序

一、输入设备

1．键盘（Keyboard）

键盘是计算机中最常用的输入设备，由按键、键盘架、编码器、键盘接口及相应控制程序等几部分组成，如图 A-1 所示。键盘通常有几十或上百个键，每个键相当于一个开关。一般微型机的键盘包括标准键盘（83 键、84 键）和扩展键盘（101 键、104 键）两种。

图 A-1　键盘

1）键盘的构成

键盘内主要由单片机、译码器和 16 行×8 列的键开关阵列这三部分组成。单片机，就是将主机的 4 个组成部分：CPU、存储器、总线及接口集成在一片硅片上。不同性能的单片机，这 4 部分的性能、容量等有较大的差别。键盘中使用的单片机通常是 8 位字长的，内含 2KB 的只读存储器（ROM）、128 字节的随机存取存储器（RAM）、2 个 8 位 I/O 接口、1 个 8 位定时/计数器及时钟发生器等。

2）IBM PC 键盘的特点和分类

IBM PC 系列键盘具有以下两个基本特点：

（1）按键开关均为无触点的电容开关。它是通过按键的上下动作，使电容量发生变化，来检测按键的断开或接通。除电容式开关外，常见的键开关还有霍尔效应式开关和触点式开关。

（2）PC 系列键盘属于非编码键盘。键盘按照键开关的类型可分为触点式和无触点式两种；从按键材料上分则有机械触点式、薄膜式和电容式；而从功能上讲，一般又将键盘分为编码键盘和非编码键盘。

对编码键盘，当有键按下，系统可以自动检测，并能提供按键的对应键值。这种键盘接口简单，使用方便，但价格较贵。

对非编码键盘，只简单提供键的行列位置（位置码或称扫描码），而按键的识别和键值的确定等工作全靠软件完成。

PC 系列键盘不是由硬件电路输出按键所对应的 ASCII 码值，而是由单片机扫描程序识别按键的当前位置，然后向键盘接口输出该键的扫描码。按键的识别、键值的确定，以及键代码存入键缓冲区等工作全部由软件完成。

目前，PC 上常用的键盘插口有三种：一是比较老式的直径 13mm 的 PC 键盘插口；第二种是最常用的直径 8mm 的 PS/2 键盘插口；第三种 USB 接口的键盘，现在也逐渐流行起来。

2．鼠标（Mouse）

鼠标也是一种常用输入设备，其功能与键盘的光标键相似。通过移动鼠标可以快速定位屏幕上的对象，是计算机图形界面交互的必用外设之一，如图 A-2 所示。

鼠标一般通过微型机中的 RS-232C 串行接口、PS/2 鼠标插口或 USB 接口与主机连接。

图 A-2　鼠标

鼠标的操作包括两种：一种是平面上的移动；另一种就是按键的按下和释放。当鼠标器在平面上移动时，通过机械或光学的方法把鼠标器移动的距离和方向转换成脉冲信号传送给计算机，计算机鼠标驱动程序将脉冲个数转换成鼠标器的水平方向和垂直方向的位移量，从而控制显示屏上光标箭头随鼠标的移动而移动。

鼠标驱动程序（Mouse Driver）是鼠标与应用程序之间的接口，属于系统软件，在装入内存后，入口地址存放在中断矢量表中，矢量码为 33H。在汇编语言程序设计中，可通过软中断指令 INT 33H 调用鼠标驱动程序中的子程序，以实现对鼠标的应用。

鼠标的分类方法很多，若按照接口类型分，可分为 5 类：PS/2 接口、串行接口、USB 接口、红外接口和无线接口。PS/2 鼠标用的是 6 针的小型圆形接口，串口鼠标用的是 9 针的 D 型接口，USB 鼠标使用 USB 接口，具有即插即用特性。红外接口鼠标用红外线与计算机进行数据传输，无线接口鼠标则通过无线电信号与计算机进行数据传输，这两种鼠标都没有连接线，故又称为摇控鼠标，使用起来较为灵活，不受连接线的限制。但红外接口鼠标使用时要正对着计算机，角度不能太大，而无线鼠标就没有这个限制。

按照不同的工作原理，鼠标又可以分为机械式、光电式和光机式。

最常见的鼠标是机械式鼠标，其底部有一个被橡胶包盖着的金属球，紧靠着橡胶球有两个相互垂直的转轴，在转轴上装着旋转编码器和相应电路。当鼠标器移动时，球便滚动，使两个转轴旋转，由编码器及相应电路可计算沿水平方向和垂直方向偏移量。这种鼠标器结构简单、价格便宜、操作方便，准确度、灵敏度差。

目前最流行的鼠标是光机式鼠标。为光学和机械混合结构。它将两个相互垂直的滚轴紧靠在橡胶球上，两个滚轴顶端各装有一个边缘开槽的光栅轮，光栅轮的两边分别装着发光二极管和光敏三极管。当鼠标器移动时，橡胶球滚动，带动滚轴及光栅轮转动，碰到栅轮的开槽时光线透过，遇到未开槽则不透光。从而使光敏三极管产生高、低电平，形成脉冲信号。光电鼠标是在鼠标底部用一个图形识别芯片时刻监视鼠标与桌面的相对移动，根据移动情况发出位移信号。这种鼠标器传送速率快，灵敏度和准确度高，但价格较贵。

对笔记本计算机，其鼠标包括内置式和外置式两种。外置式鼠标与普通台式机鼠标完全相同。内置式鼠标则与机器合为一体，在工作原理上有指点杆式、触摸屏式和轨迹球式。

鼠标器最重要的参数是分辨率。它以 dpi（像素/英寸）为单位。表示鼠标移动 1in 所通过的像素数。一般鼠标器的分辨率为 150～200dpi，高的可达 300～400dpi，若屏幕分辨率为 640×480dpi 时，鼠标器只要移动 1in，则对应屏幕 300～400 像素位置，基本遍历屏幕的 2/3。因此，鼠标的分辨率越高，鼠标器移动距离就越短。

二、输出设备

输出设备用于接收或传输计算机的处理结果。最基本和最常用的就是显示器和打印机。

1．显示器

显示器的作用是将主机输出的电信号经一系列处理后转换成光信号，并最终将文字、图形显示出来。常用的显示器有阴极射线管监视器（CRT）和液晶显示器（LCD）两种。图 A-3 所示为 LCD 显示器。

图 A-3　LCD 显示器

CRT 显示器分为荫罩式和电压穿透式。目前已基本退出市场。

LCD 显示器采用的技术主要有两种：有源矩阵和无源矩阵。

有源矩阵显示器又称为薄膜晶体管液晶显示器（TFT）。它的每一个像素点都用一个薄膜晶体管来控制液晶的透光率，优点是色彩鲜艳，视角宽，图像质量高，响应速度快。但其成品率低，从而导致价格比较昂贵。

无源矩阵显示器用电阻来代替有源晶体管，制造较为容易。它和有源矩阵相比的最大优势就是价格低。其缺点是色彩饱和度较差，图像不够清晰，对比度也较低，视角较窄，响应速度慢。

LCD 显示器与 CRT 显示器相比较，其特点是外尺寸相同时可视面积更大，体积小（薄），外形美观，图形清晰，不存在刷新频率和画面闪烁的问题，但价格比较昂贵，分辨率较低。

2．打印机

打印机也是计算机系统的标准输出设备之一，如图 A-4 所示。它与主机之间的数据传送方式有并行的，也可以是串行的。目前，大多数打印机采用并行数据传送方式，即通过并行接口与主机连接。对部分串行打印机，则通过主机的串行口连接。

图 A-4　打印机

打印机的种类很多。按照打印原理，可分为击打式打印机和非击打式打印机。

击打式打印机是用机械方法，使打印针或字符锤击打色带，在打印纸上印出字符。非击打式打印机是通过激光、喷墨、热升华、热敏等方式将字符印在打印纸上。

1）打印机工作方式

同显示器一样，打印机在微机系统中的工作方式也可按其从主机接收的数据类型分为字符方式和图形方式。

字符方式，是指主机在发送打印数据时，只传送字符的 ASCII 码，打印机根据收到的 ASCII 码从字模 ROM 中取出相应的字符点阵信息，最后用机械、光学或加热的方法打印到纸上。汉字的打印也可以在字符方式下进行，这要以打印机内部具备全部汉字字模为前提。字符方式可以获得较快的打印速度，是当前西文打印中最常用的方法，中文打印如采用这种方式，打印机的成本就要相应提高。字符方式不能用于图形打印。

在图形方式下，主机所传送的不是字符代码，而是经过软件编辑的图形像素信息。图形方式既可以打印西文字符，也可以打印汉字或任意的图像。

在微机系统中，上述两种打印方式往往是共存的，到底使用哪一种方式要视具体情况而定。有时，用户可用键盘输入命令或通过程序中给定的指令来选择其一，有时由系统规定而不能改变。

打印机通过接口与主机相连，该接口又称为打印机控制器或适配器。它可以是一块独立的接口卡，也可以集成在主板上（现代微型机的主板上几乎无一例外地都集成了打印机的接口）。它们通过标准的 25 芯插头插座相连接。

在 CPU 与打印机进行数据传送时，首先要由接口向打印机提供“选择输入”控制信号，打印机在此信号控制下，才能接收数据及其他控制信号；同时，打印机要向接口送上有效的“打印机选中”状态信号，表示打印机已加电工作。之后，CPU 通过接口向打印机输出数据（字符）。这种输出是一个字节一个字节进行的。每一次“选通”，输出一个字节到打印机内部的缓冲存储器，直到全部数据传送完毕。许多打印机还可提供“忙”、“纸尽”等状态信号，以停止主机做相应的处理。

2）主要性能指标

衡量打印机性能的主要指标包括以下几个方面：

- **分辨率**。分辨率用 DPI 表示，即每英寸打印点数，它是衡量打印质量的重要指标。不同类型的打印机其打印质量也不同。针式打印机的分辨率较低，一般为 180～360dpi。喷墨打印机分辨率一般为 300～1440DPI。激光打印机的分辨率为 300～2880dpi。
- **打印速度**。针式打印机的速度用每秒打印字符数 CPS 表示。打印速度在不同的字体和文种下差别较大。针式打印机的打印速度由于受机械运动的影响，在印刷体方式下一般不超过 100CPS，在草稿方式下可以达到 200CPS。喷墨打印机和激光打印机都属于页式打印机（即计算机输出完一整页的内容，打印机才开始打印），打印速度以每分钟打印页数（PPM）表示，一般在几 PPM 到几十 PPM 之间。
- **汉字打印、中西文字库及打印字体**。能否打印汉字是衡量打印机性能的一项重要指标。有无中文字库对打印机的打印速度影响很大。另外，打印字体也是一个影响速度的因素。目前，针式打印机打印汉字字体最少为 4 种（宋体、仿宋体、楷体、黑体），打印各类英文、数字字符共 5～10 种；喷墨打印机打印的西文字体有 6～8 种，中文 3 种以上；激光打印机有 3 种中文字体（宋、楷、黑）及各种英文字体。以上均指打印机自带字库的情况，若使用图形打印方式，则打印字体仅与主机支持的字体数量有关。
- **打印缓冲存储器**。打印机设置较大的缓冲存储器是为了满足高速打印和打印大型文件的需要。缓冲存储器的大小将影响打印速度。针式打印机的缓冲存储器一般为 16KB。喷墨打印机和激光打印机的缓冲存储器因在图形方式下要存储大量的图形点阵信息，并且是整页装入，其缓冲存储器较大，通常容量可达 4～16MB。
- **打印幅面**。打印幅面问题是用户直接关心的问题。对针式打印机，规格有两种：80 列和 132 列即每行可打印 80 个或 132 个字符。对非击打式打印机，幅面一般为 A4、A3 和 B4。
- **接口类型**。打印机的接口类型主要有三种：并行接口、串行接口和 USB 接口。并行接口应用最广泛，所以，人们往往把计算机上的并行接口又称为打印机接口。

3）几种常见打印机

目前，市场上常见的打印机有点阵式、喷墨和激光打印机三种。点阵式打印机现在主

要用于银行、税务等部门的票据类打印，喷墨和激光打印机则因其打印性能、效果等方面的优势而越来越得到更广泛的应用。

三、设备驱动程序

1．设备驱动程序的一般概念

设备驱动程序是对连接到计算机系统的设备进行控制驱动以使其正常工作的一种软件。在当前流行的几乎所有的操作系统中，设备驱动程序都被认为是最核心的一类部件，处于操作系统的最深层，故要重写这些驱动程序是很困难的。

有些用户可能会遇到这样的现象：将光盘放入光盘驱动器后，机器却找不到光驱，这是为什么呢？原因很简单，就是光驱驱动程序没有安装。我们在平时使用计算机时，不仅是光驱需要安装驱动程序，还有如声卡、显示卡、解压卡、网卡、MODEM、激光或喷墨打印机，以及前文提到的可移动硬盘等都需要安装驱动程序。实际上，计算机中所有的硬件都需要驱动程序，但为什么使用键盘、鼠标、软驱和硬盘就不用安装驱动程序呢？这是因为这些设备的接口规范已经标准化，不需再作任何修改就能在各种环境下使用，它们的设备驱动程序已被固化在 BIOS 中作为标准的驱动程序供操作系统或应用程序使用（也可以说它们在计算机生产过程中已经被预先安装到了系统中）。

由上述可知，驱动程序是通过一组预先定义好的软件接口为操作系统或应用程序提供控制硬件能力的一组软件程序。它的好处有两点：一是由于有了驱动程序这一软件层次，使操作系统或应用程序就没有必要关心硬件设备的具体操作细节，大大降低了软件的开发难度和软件的复杂程度；二是增强了软件的兼容性，例如，更换了设备后，只要相应地更换驱动程序即可，而无须将整个操作系统或应用程序都换掉。当然，如果在应用程序中不通过设备驱动程序而直接访问硬件也是可以的，但这会带来兼容性问题，也就是说，硬件变化后必须要重新编写全部应用程序。

由于不同的操作系统对硬件的管理、控制、使用的方式方法都存在一定的差异，所以，即使是同一件硬件设备，当其在不同的操作系统中使用时，也需要各个系统中专门设计的驱动程序来支持。因此，在硬件使用前，查找硬件附带的驱动程序，查阅相关驱动程序的安装和配置方法是一项十分重要的工作。

2．硬件设备的“即插即用”概念

微软公司在开发 Windows 95 时，为解决用户对外部设备硬件参数设置的困扰而开发了一项新的功能：即插即用（Plug and Play，PnP）。这是一项用于自动处理 PC 硬件设备安装的工业标准，由 Intel 和 Microsoft 两大公司联合制定。

用户需要安装新的硬件时，往往要考虑到该设备所使用的各种资源，以避免设备之间因竞争而出现冲突（如两个设备可能占用同样的中断号、I/O 地址等）。这是一项很麻烦的工作，而有了“即插即用”功能，就使得硬件设备的安装大大简化了，用户无须再选择如何跳线，也不必使用软件配置程序，一切都可由操作系统代替完成。但要做到“即插即用”，对所安装的硬件就有一定的要求，即必须是符合 PnP 规范的，否则无法做到即插即用。“即插即用”是 Windows 95 及以后的操作系统最显著特征之一，基于 Intel 体系结构的其他微机操作系统目前尚不具备该特性。

即插即用特性还需要主板具有 PnP 功能，这样在系统启动时由 BIOS 自动读取提供具有 PnP 功能的接口卡的设定参数，自动分配各项资源，并将分配后的设定参数存入主机板上的闪速存储器（Flash Memory），再由操作系统从主板闪存读取编排后的 PnP 界面上相关设定参数，从而避免以往因 I/O 地址相互冲突所造成的困扰，使整个计算机在执行各种程序时有效地发挥系统功能。

“即插即用”计算机系统的具体内容如下：

（1）支持“即插即用”的 BIOS。

PnP BIOS 提供基本指令集，用于确定在系统开机自检（POST）时所需要的最基本设备。这些设备至少包括显示器、键盘、磁盘驱动器等。

（2）“即插即用”操作系统。

Windows95 是第一个支持 PnP 的操作系统，之后的 Windows 系统也都支持即插即用。

（3）“即插即用”硬件。

“即插即用”硬件是指由 PnP 操作系统自动配置的一组 PC 硬件设备。PnP 也同样支持打印机、调制解调器、串行口和并行口等，基于 ISA 和 EISA 的适配卡则需要进行适当的修改。

（4）“即插即用”设备驱动程序。

Microsoft 提供的设备驱动程序支持基本 PnP 设备，如 IDE 硬盘、CD-ROM 等。

附录 B　标准 ASCII 码表

ASCII 码表

	列	0	1	2	3	4	5	6	7
行	高位 / 低位	000	001	010	011	100	101	110	111
0	0000	NUL	DLE	SP	0	@	P	、	p
1	0001	SOH	DC1	!	1	A	Q	a	q
2	0010	STX	DC2	“	2	B	R	b	r
3	0011	ETX	DC3	#	3	C	S	c	s
4	0100	EOT	DC4	$	4	D	T	d	t
5	0101	ENQ	NAK	%	5	E	U	e	u
6	0110	ACK	SYN	&	6	F	V	f	v
7	0111	BEL	ETB	‘	7	G	W	g	w
8	1000	BS	CAN	(	8	H	X	h	x
9	1001	HT	EM	)	9	I	Y	i	y
A	1010	LF	SUB	*	:	J	Z	j	z
B	1011	VT	ESC	+	;	K	[	k	{
C	1100	FF	FS	,	<	L	\	l	|
D	1101	CR	GS	—	=	M	]	m	}
E	1110	SO	RS	.	>	N	Ω	n	～
F	1111	SI	US	/	?	O	_	o	DEL

注：表中的 00H～1FH 及 7FH 为控制符，不可显示；其余的为可显示字符。

ASCII 码表中控制符号的定义

NUL	Null　空白	DLE	Data Link Escape　转义
SOH	Start Of Heading　标题开始	DC1	Device Control 1　设备控制 1
STX	Start Of Text　正文开始	DC2	Device Control 2　设备控制 2
ETX	End Of Text　正文结束	DC3	Device Control 3　设备控制 3
EOT	End Of Transmit　传输结束	DC4	Device Control 4　设备控制 4
ENQ	Enquiry　询问	NAK	Negative Acknowledge　否定
ACK	Acknowledge　承认	SYN	Synchronize　同步
BEL	Bell　响铃	ETB	End of Transmitted Block　信息组结束
BS	Backspace　退格	CAN	Cancel　作废
HT	Horizontal Tab　横向制表	EM	End of Medium　纸尽
LF	Line Feed　换行	SUB	Substitute　取代
VT	Vertical Tab　纵向制表	ESC	Escape　换码
FF	Form Feed　换页	FS	File Separator　文件分隔符
CR	Carriage Return　回车	GS	Group Separator　组分隔符
SO	Shift Out　移出	RS	Record Separator　记录分隔符
SI	Shift In　移入	US	Unit Separator　单元分隔符
SP	Space　空格	DEL	Delete　删除

附录 C　声音、图像信息的数字化

一、声音信息的数字化

1．声音的基本参数

声音是通过空气传播的一种连续的波，称为声波（Sound Wave）。当声波到达人耳鼓膜时而使人感到的压力变化，就是声音（Sound）。简言之，声音是连续变化的波形，这种连续性体现为幅值大小是连续的，可以是实数范围内的任意值；在时间上是连续的，没有间断点。我们将这种在时间和幅值上都连续变化的信号称为模拟信号。相应的，将时间和幅值都不连续的信号称为离散信号。

要使声音信号能够被计算机处理，首先需要对其进行数字化，即，将时间和幅值均连续变化的模拟声音信号，通过采样（Sampling）和量化（Measuring），转换为时间和幅值均不连续的离散信号，这种离散的声音信号称为数字音频信号，也就是计算机能够存储和处理的信号。

表征声音的基本参数如下：

- **幅度**（Amplitude）。指声音的大小或强弱程度，幅度越大，表示声音越高。
- **频率**（Frequency）。指信号每秒钟变化的次数，用赫兹（Hz）表示。频率越高，声音听上去就越“尖锐”。低于或高于一定频率后的声音我们就听不到了。
- **带宽**（Band Width）。声音信号的频率范围，如高保真声音的频率范围为 10～20 000Hz，它的带宽约为 20kHz。
- **亚音信号**（Subsonic）。频率小于 20Hz 的人听不到的声音信号。人们对声音的感知不仅与声音的幅度有关，还与声音的频率有关。中频或高频中可感知的相同的音量在处于低频时需要更高的能量来传递。例如，大气压的变化周期很长，以小时或天数计算，一般人不容易感到这种气压信号的变化，更听不到这种变化。
- **音频信号**（Audio）。频率范围为 20Hz～20kHz 的、人能够听到的声音信号。
- **超音频信号**（Supersonic）。又称超声波（Ultrasonic）。高于 20kHz 的信号。

计算机中处理的声音信号主要是音频信号，包括音乐、话音、风声、雨声、鸟叫声和机器声等。音频信号的带宽（频率范围）越宽，声音的质量（音质）就越好。

2．声音信号的数字化

要使连续变化的声音信号（模拟信号）能够为计算机处理，必须要将其转变为离散（不连续）的数字信号。将时间和幅值均连续变化的模拟声音信号转换为在时间和幅值上均离散的数字信号的过程称为声音信号的数字化。这是声音信号进入计算机的第一步，数字化的主要工作就是采样（Sampling）和量化（Measuring）。声音的采样和量化如图 C-1 所示。

采样是指定期在某些特定的时刻对模拟信号进行测量。采样的结果是得到在时间上离

散，但幅值上连续变化（幅值可以是任意一个实数值）的离散时间信号（Discrete Amplitude Signal）。

对这种连续幅值的离散时间信号，计算机是无法进行处理的。还需要将信号幅度的取值数目加以限定，将任意实数值的幅度值由有限个数值组成，使信号不仅在时间上同时也在幅值上离散的离散幅度信号（Discrete Amplitude Signal）。例如，设输入电压的范围是0.0～0.7V，而它的取值仅限定在 0V，0.lV，0.2V，…，0.7V 共 8 个值。如果采样得到的幅度值是 0.123V，则近似取值为 0.1V，如果采样得到的幅度值是 0.271V，它的取值就近似为 0.3V，这种数值就称为离散数值，对幅值进行限定和近似的过程称为量化。把时间和幅度都用离散数字表示的信号就称为数字信号（Digital Signal），如图 F5（b）所示。

对声音的数字化实质上就是采样和量化。只有将连续变化的模拟声音转换为离散的数字声音频信号，计算机才能处理和存储。可以想象，在一个规定的时间里对模拟声音采样的次数越多，对原始信号的反映（还原）就会越准确，近似度就越好。当然，采样的次数越多，所得到的数据量就越多，需要占用的存储空间就越大。

我们将单位时间内的采样次数称为采样频率（Sampling Frequence）。根据奈奎斯特理论（Nyqust Theory）：如果采样频率不低于信号最高频率的 2 倍，就能把以数字表达的声音还原成原来的声音。例如，话音信号的最高频率为 3400Hz，采样频率至少应为 6800Hz 才能正确还原（在实际应用中，话音信号的采样频率规定为 8000Hz）。对于一般音频信号，最高频率为 20kHz，采样频率在 40kHz 以上时就能无失真还原出原来的声音。

除了采样频率的要求外，数字化声音的不失真还原还与幅值的量化级别有关。量化级别越多，越能反映不同的声音。例如，若只用 1 位二进制码表示声音的量化级别，则只能是有声和无声两种状态；如果用 8 位二进制数表示量化级别，就可以有 256 种幅值。用于表示量化级别的二进制数的位数，称为采样精度（Sampling Precision），又称为样本位数或位深度。对 8 位二进制数表示的声音样本，有 256 种不同的幅值，它的精度是输入信号的 1/256。

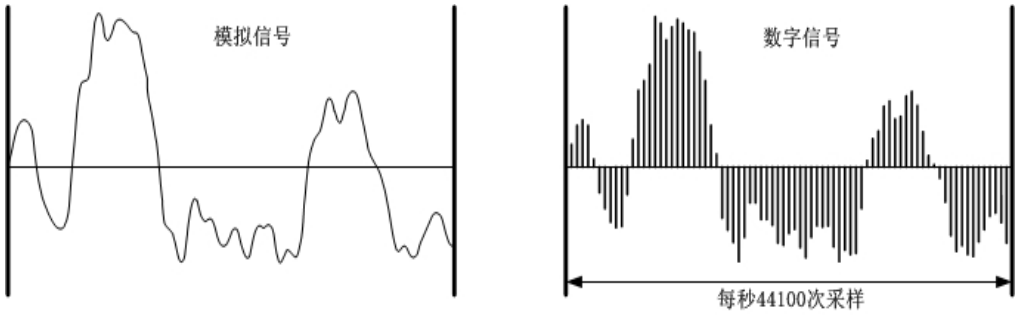


图 C-1　声音的采样和量化

采样频率越高，样本位数越多，声音的还原性越好，质量越高，所占用的存储空间也越大。一个声音文件的大小可用下式计算：

声音文件的数据量 = 采样频率（Hz）×样本位数（bit）×声道数×时间（s）

例 C-1　设采样频率为 16kHz，样本位数为 8 位，分别计算 1min 的单声道和双声道声音文件的数据量。

1min 单声道数据量=16×8×60=7680Kb=960KB

1min 双声道数据量=16×8×60×2=15360Kb=1920KB

可以看出，声音文件的数据量是比较大的。为了节省存储空间，对不需高品质音效的应用程序可以使用较低的采样速率。以 Windows XP 中的录音机程序为例，如果设定语音采样速率为 8kHz，采样精度为 8 位，单声道，则文件大小只有以 44.1kHz、16 位、双声道录制的相似声音文件的 1/22。当然，这样的声音质量比较低，但在录制话音信号时（如英文阅读）已能满足要求。

二、图像信息的数字化

1．图像的数字化

图像是在二维空间坐标上连续变化的函数，连续图像的数字化过程是空间和幅值的离散化。将空间连续坐标（x, y）的离散化称为图像的采样（Image Sampling）；幅值 f(x, y)的离散化称为整量。

采样是将一幅图像变换为 f(x, y)坐标中的一个个点，称为像素点。每一个像素点具有颜色空间中的某一种颜色（灰度值）。

采样所得到的像素点的灰度值是连续的（如同采样后的声音信号），为便于处理还必须进行整量。整量是用有限二进制数位来表示某个像素点的灰度值，所用的二进制数位越长，可以表示的灰度等级就越多。如果仅用一位二进制码表示像素点的灰度，该像素点就只有“黑”、“白”两种颜色；若用 4 位二进制码来表示，则该像素点就可以有 16 种不同的颜色（或由黑到白 16 种不同的灰度等级），相应的图像称为 16 色图像。

将一幅连续图像按一定顺序在 x 和 y 方向进行等间隔采样，就将图像变换为 N×N 个像素点组成的数组，再对这些像素点的灰度用等间隔进行整量，就得到了一幅 N×N 的数字图像（Digital Image）。例如，对图 1-15 所示的黑白二色图像，在横向和纵向各取 10 个点进行采样，可以得到 10×10 个数值，由于只有黑白两种颜色，故每个点的颜色（灰度）只需用 1 位二进制码表示，即二级量化（如用“0”表示黑，用“1”表示白）。这样，就得到 10×10 个取值为“0”或“1”数据，将这些数据按采样的行列位置排列成如图 1-16 的形式，这就是一幅二值图像在计算机中的表示。

数字图像中表示每个像素颜色所使用的二进制位数称为像素深度（Pixel Depth）或位深度。像素深度越大，图像能表示颜色数越多，色彩越丰富逼真，占用的存储空间越大。常见的像素深度有 1 位、4 位、8 位和 24 位，分别用来表示黑白图像、16 色或 16 级灰度图像、256 色（或 256 级灰度）图像和真彩色（2^{24} 种颜色）图像。

2．图像的主要性能参数

一幅图像的采样点数称为图像分辨率（Image Resolution），用点的“行数×列数”表示。如果一幅图像只取一个采样点，只得到一个数据，量化后这幅数字图像就只有一种颜色。对相同尺幅的图像，采样的点数越多，图像的分辨率就越高，所得数字图像看上去就越逼真，越“细腻”。相反，图像显得越粗糙。例如，图像分辨率为 640×480 的数码像机拍摄的照片就远比分辨率为 1128×764 相机拍摄的照片差。

图像分辨率是组成数字图像的像素数，在用扫描仪扫描图像时，还涉及到另外一种分

辨率，称为扫描分辨率（Scanning Resolution）。扫描分辨率用每英寸所含像素点数（dots per inch，dpi）表示，用于使不同尺寸的图像获得相同的扫描精度。

扫描分辨率和图像分辨率不同，扫描分辨率是采样时，单位尺寸内采样的点数；图像分辨率是组成数字图像的像素数。例如，用 200DPI 来扫描一幅 6"×8"的图像,得到一幅 1 200×1 600 个像素的数字图像。

图像文件的大小由图像分辨率和像素深度。一幅位图图像文件的大小可由下式估算：

图像分辨率×像素深度

例 C-2　计算一幅图像分辨率为 640×480 的真彩色图像（位深度 24 位）的文件大小。

图像文件数据量= 640×480×24=3 732 800（bits）

= 921 600B（Bytes）

由上例可以看出，图像的分辨率越高，样本位数越长，图像文件占用的存储空间就越大，其传输需要的时间也就越长。如果在家里从互联网下载一个 640×480 大小的 256 色位图要花费半分钟或更长时间，而下载 16 色同样大小的图像文件则可以减少 1/2 的时间。

数字图像的视觉效果与图像输出设备有关，图像在屏幕上的显示尺幅称为图像的显示分辨率（Display Resolution）。分辨率低的图像可以以高的分辨率显示，分辨率高的图像也可以以低的分辨率显示，但只要不是以图像的正常分辨率显示图像，都会引起图像的失真。所以，使用图像时应按需要设置图像的分辨率和像素深度。

参考文献

[1] 王伟．计算机科学前沿技术．北京：清华大学出版社，2012

[2] 张立昂．可计算性与计算复杂性导引．北京：北京大学出版社，2011

[3] 冯博琴，吴宁．微型计算机原理与接口技术（第 3 版）．北京：清华大学出版社，2011

[4] 汤子瀛，哲凤屏，汤小丹．计算机操作系统．陕西：西安电子科技大学出版社，1996

[5] 张江．图灵机与计算问题．http://www.doc88.com/p-984356866205.html

[6] 谢希仁．计算机网络（第 5 版）．北京：电子工业出版社，2008

[7] 冯博琴，陈文革．计算机网络（第 2 版）．北京：高等教育出版社，2008

[8] 陈文革．计算机网络技术经典实验案例集．北京：高等教育出版社，2012

[9] 百度百科．http://baike.baidu.com/

[10] 维基百科．http://zh.wikipedia.org/wiki/Wikipedia

[11] 严蔚敏，吴伟民．数据结构．北京：清华大学出版社，1992

[12] Paul Deitel，张思宇，译．Visual C# 2010 大学教程（第 4 版）．北京：电子工业出版社，2011

内容简介

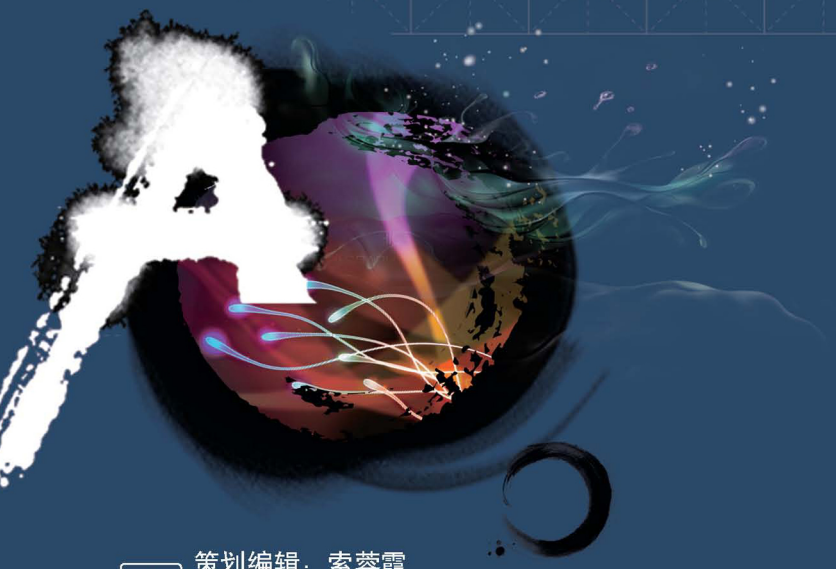
本书是国家精品课程“大学计算机基础”的主教材，是在《大学计算机基础》（2011年8月第1版）的基础上修订而成的，此次再版，融入了两年教学实践的新体会，同时，对部分内容进行了调整和修改。

本书以“计算思维能力”培养为主线，强调“计算机基本工作原理”的理解和“问题求解思路”的建立。在组织架构上主要分为四个部分，共8章，主要内容包括计算与可计算性理论简述（引言部分）、计算机基础知识、微型计算机系统、计算机网络及应用、Visual Basic程序设计、数据结构、算法分析与设计和综合案例。

本书各章均在起始处给出了该章的引言及教学目的，以供读者学习时参考。同时，还配有大量图示和例题，以便于对内容的理解。为方便教学，本书还免费提供电子课件，可以登录华信教育资源网（www.hxedu.com.cn）注册下载。

本书可作为普通高等学校非计算机专业“大学计算机基础”课程的教材，适用学时为48～64学时。目录中带有“*”的章节为可选内容，可根据具体情况选择讲授或作为课外开放性学习使用。

提升学生“知识—能力—素质”	体现“基础—技术—应用”内容
把握教学“难度—深度—强度”	提供“教材—教辅—课件”支持



策划编辑：索蓉霞
责任编辑：郝黎明
封面设计：张　昱

ISBN 978-7-121-15529-1

定价：35.00 元